Universidad
Carlos III de Madrid

Departamento de ING. SISTEMAS Y AUTOMÁTICA

PROYECTO FIN DE CARRERA

INTEGRACIÓN DE OPENGL EN QT

Autor: Douglas Humberto Guzmán Centanaro.

Tutor: Arturo de la Esacalera

Leganés, noviembre del 2011

Título: Integración de OpenGl en Qt
Autor: Douglas Humberto Guzman Centanaro
Director: Arutor de la Escalera

EL TRIBUNAL

Presidente: Jose Maria Armingol moreno

Vocal: Fernando García Fernandez

Secretario: Jorge Ardila Rey

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 18 de noviembre del 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Agradezco a mi tutor Arturo de la Escalera por haberme dado la oportunidad de trabajar en este proyecto y estar presente de manera constante en realización del mismo.

Resumen

El objetivo de este proyecto ha sido el diseño en tres dimensiones de un coche Nissan "Note" y un carro de golf, para que sirvan de interface gráfica entre el usuario y el hardware, ambos vehículos en fase de automatización. Los diseños estan dotados de movimiento en las tres dimensiones del sistema de coordenadas, según desee el usuario mediante el teclado o el ratón del ordenador.

Para el desarrollo de los gráficos ha sido necesario integrar la biblioteca gráfica gratuita OpenGL en el programa gratuito QtCreator de Nokia, en un ordenador bajo el sistema operativo Windows Vista.

Hay que recalcar la importancia de usar la biblioteca gráfica OpenGl ya que esta biblioteca pérmite que el código sea portable a cualquier otro entorno de desarrollo integrado(*IDE*), y por lo tanto es independiente del mis

ÍNDICE GENERAL

Capítulo 1

INTRODUCCIÓN: A LOS GRÁFICOS 3D Y OPENGL.

1.1 Introducción

La función principal de este proyecto es el desarrollo de dos vehículos en tres dimensiones , con traslación en los ejes x, y, z mediante las teclas direccionales del teclado u el ratón del ordenador. Los diseños servirán de interfaz gráfico para los vehículos reales que se están automatizando como son el "Nissan NOTE" y el coche de golf.

El objetivo fundamental es crear los vehículos mediante la integración de la biblioteca gráfica "OpenGL" en el programa Qtcreator.

- Para desarrollar el proyecto se ha hecho uso de un ordenador portátil DELL inspiron.1720, Intel Core 2 Duo T7100 1.8 GHz 4MB L2 Cache, 667 MHz FSB, tarjeta gráfica NVIDIA Geforce 8600M GT - 256 MB GDDR2 VRAM, Taktrate Chip/VRAM: 475/400MHz.

- Se ha instalado en el ordenador el paquete de software gratuito Qtcreator para windows vista desde la página web : http://qt.nokia.com/downloads/ .

- Para el diseño del coche "NISSAN NOTE" se ha recurrido a descargar fotos desde la página web oficial de "Nissan". http://www.nissan.es, y para el diseño del carro de golf se ha hecho uso del buscador google. www.google.es . A partir de las imágenes se ha empezado a tomar cotas, alturas, alejamientos, y a partir de allí convertirlos en coordenadas x, y, z.

1.2 Objetivos

El objetivo fundamental de la tesis es el de servir de guía a futuras personas que deseen crear gráficos con una biblioteca libre como es OpenGl (*Open Graphics Library*) en un programa libre como es QtCreator.

1.3 Fases del desarrollo

- Lo primero que hay que hacer es tener instalado de forma correcta la versión de QtCreator en el ordenador. Una vez instalado se procede a integrar el módulo de OpenGL en dicho programa.
- A partir de allí en QtCreator hay que crear y configurar la ventana necesaria para poder desarrollar el diseño que se desea mediante el uso de las herramientas proporcionadas con OpenGl.
- Posteriormente se realiza el proceso creativo del diseño en el plano cartesiano y luego se traspasa dicho diseño al código de C++ mediante el uso de las herramientas de OpenGL.

1.4 Medios empleados

Para desarrollar el proyecto y la tesis se ha hecho uso de multitud de tutoriales buscados en Internet, páginas web oficiales, como son www.qtcreator.es y www.opengle.org , además de un gran libro como es la biblia de OpenGl . Todos estos están citados en la bibliografía utilizada.

1.5 Estructura de la memoria

- El objetivo de este primer capitulo será el de dar una idea mas clara y de los principales conceptos de los gráficos en tres dimensiones con los que convivimos día a día. Además se introducirá el concepto de que es "OpenGL" y de cómo funciona.

- En el capítulo posterior se hablará del programa de Nokia "Qtcreator", sobre sus herramientas y cómo se integra el módulo de OpenGl en dicho programa.

- En el capítulo siguiente se desarrollará más a fondo pero sin meterse de manera excesiva en la herramientas que proporciona OpenGl y como se ha hecho uso de alguna de ellas.

- En posteriormente se introducirá los conceptos de color y material.

- Después se hará mención del uso de texturas en los diseños

- Tras haber mostrado todas las herramientas necesarias para el desarrollo del diseños, se mostrarán imágenes con distintas fases de su creación.

1.6 USOS COMUNES PARA GRÁFICOS DE 3D:

Las aplicaciones para gráficos 3D en el PC son casi ilimitadas quizá el uso más común de hoy en día sean los juegos por ordenador. Muy pocos títulos actuales no requieren una tarjeta gráfica 3D en el PC para poder jugar. Los gráficos 3D siempre han sido populares para las aplicaciones de visualización e ingeniería científica,pero la aparción de hardware mas barato, ha favorecido estas aplicaciones más que nunca, como se muestra en la (FIGURA 1) **.** Las aplicaciones comerciales también se estan aprovechando de las nueva disponibilidad de hardware para incorporar gráficos comerciales mas complejos y técnicas de visualización de exploración de bases de datos.

ECOGRAFIA EN 3D

(FIGURA 1)

Normalmente, se diseña modelos y escenas y un trazador de rayos procesa la definición para producir una imagen 3D de alta calidad. El proceso típico es que alguna aplicación de modelado 3D use gráficos en tiempo real para interactuar con el artista que va a crear el contenido. A continuación los fotogramas se envían a otra aplicación (el trazador de rayos) que interpreta la imagen. Por ejemplo interpretación de un solo fotograma para una película como Toy Story puede tardar horas en un ordenador muy rápido. El proceso de interpretación y guardado de muchos miles fotogramas genera una secuencia animada para su reproducción. Aunque pueda parecer que la reproducción se realiza en tiempo real, el contenido no es interactivo, por lo que no se considera una reproducción en tiempo real, sino una reproducción previa la interpretación.(FIGURA 2)

videojuego insertado en un teléfono

(FIGURA 2)

Es muy importante dentro de los gráficos en computación la capacidad de conseguir movimiento a partir de una secuencia de fotogramas o “frames”. La animación es fundamental para un simulador de vuelo, una aplicación de mecánica industrial o un juego.

En el momento en que le ojo humano percibe mas de 24 frames en un segundo, el cerebro lo interpreta como movimiento real. En ese momento , cualquier proyector comercial es capaz de alcanzar frecuencias de 60 frames/s o mas. Evidentemente 60 fps (frames por segundo) será mas suave (por tanto mas real) que 30 fps. Según[2].

La clave para que la animación funciones es que cada frame este competo (haya terminado de renderizar) cuando sea mostrado por pantalla. Según [2].

1.7 ¿QUÉ ES OPENGL?

OpenGl se define estrictamente como “una interfaz de software para hardware gráfico ”. básicamente se trata de una biblioteca de modelado y graficos 3D que se puede exportar muy rápidamente. Con OpenGl se puede crear gráfico reales y bonitos con una calidad visual cercana a la de un trazador de rayos. La gran ventaja de usar OpenGl es que es mucho más rápido que un trazador de rayos.

OpenGl no es un lenguaje de programación como C o C++. Es mas parecido a una biblioteca de C en tiempo de ejecución que proporciona una funcionalidad empaquetada previamente. En realidad no existe nada parecido a un “programa en OpenGl”, sino un programa que el desarrollador ha escrito para usar OpenGL como una de sus interfaces de programación de aplicaciones (API, *Application Programming Interfaces*). Según [1] . Es un motor 3D cuyas rutinas están integradas en tarjetas gráficas 3D.

Fue desarrollado por “*Sillicon Graphics, Inc*.” (SGI) con el afán de hacer un estándar de representación en 3D. según[2].

OpenGL se usa para una amplia variedad de fines, desde las aplicaciones CAD de ingeniería y arquitectura a videojuegos, películas. La presentación de una API 3D como norma industrial para un sistema operativo presente masivamente en el mercado como Microsoft Windows tiene algunas repercusiones importantes.

A medida que se popularizan el hardware de aceleración y los microprocesadores rápidos para PC, los gráficos 3D serán, en poco tiempo, un componente típico de las aplicaciones mercantiles y de consumo, no únicamente reservadas a los juegos o a las aplicaciones científicas.

Su antecesora fue GL, de “*Silicon Graphics”*. "*Iris GL*" fue la primera API de programación para las estaciones gráficas IRIS de alto rendimiento. según[1]

Estos ordenadores eran algo más que máquinas de propósito general, pues tenían un hardware especialmente optimizado para la visualización de gráficos, y cuando SGI trató de trasladar IRIS GL a otras plataformas hardware, tuvieron serios problemas.

OpenGL es el resultado de los esfuerzos de SGI para mejorar la portabilidad de IRIS GL. El nuevo lenguaje debía ofrecer la potencia de GL y ser a la vez un lenguaje más "abierto" (open), permitiendo una fácil adaptación a otras plataformas hardware y sistemas operativos.

El 1 de julio de 1992 se presentó la versión 1.0 de la especificación OpenGL. Mientras SGI realizaba una serie de demostraciones en unas jornadas de desarrolladores de Win32, a la vez comenzó a trabajar con Microsoft para incluir OpenGL en una futura versión de Windows NT. Según [1].

Al ser una norma abierta, todas las mejoras de OpenGL son decididas por la Plataforma para Revisiones de la Arquitectura OpenGl (*ARB*), que se reúne dos veces al año.
Los miembros de la ARB participan frecuentemente en el grupo de noticias de Internet (***comp.graphics.api.opengl***), donde se pueden airear cuestiones y recomendaciones. según [1]

1.8 CARACTERÍSTICAS

OpenGL es un lenguaje procedimental.

En lugar de diseñar la escena y cómo debe aparecer, el programador describe los pasos necesarios para conseguir una cierta apariencia o efecto, que implican llamadas a una API altamente portátil que incluye 120 comandos y funciones.

No incluye funciones para la gestión de ventanas, interactividad con el usuario ni manejo de ficheros.

Cada entorno anfitrión (como Microsoft Windows) tiene sus propias funciones para este propósito y es responsable de implementar alguna manera de facilitar a OpenGL la facultad de dibujar en una ventana o mapa de bits.

La implementación genérica OpenGL de Microsoft tiene ciertas limitaciones.

A menos que esté específicamente soportada por hardware, la implementación genérica tiene las siguientes limitaciones:

- No hay ningún soporte directo para imprimir gráficos OpenGL a una impresora monocroma o de color con menos de 4 planos de bits de color (16 colores).

- No se soportan paletas hardware para varias ventanas, en su lugar, Windows tiene una única paleta hardware que debe ser arbitrada entre múltiples aplicaciones.
- Algunas prestaciones de OpenGL no están implementadas, incluidas las imágenes estereoscópicas, buffers auxiliares y planos de bits alpha. Las aplicaciones deben comprobar su disponibilidad por hardware antes de hacer uso de ellas.

[5]

1.9 LICENCIAS Y CONFORMIDAD

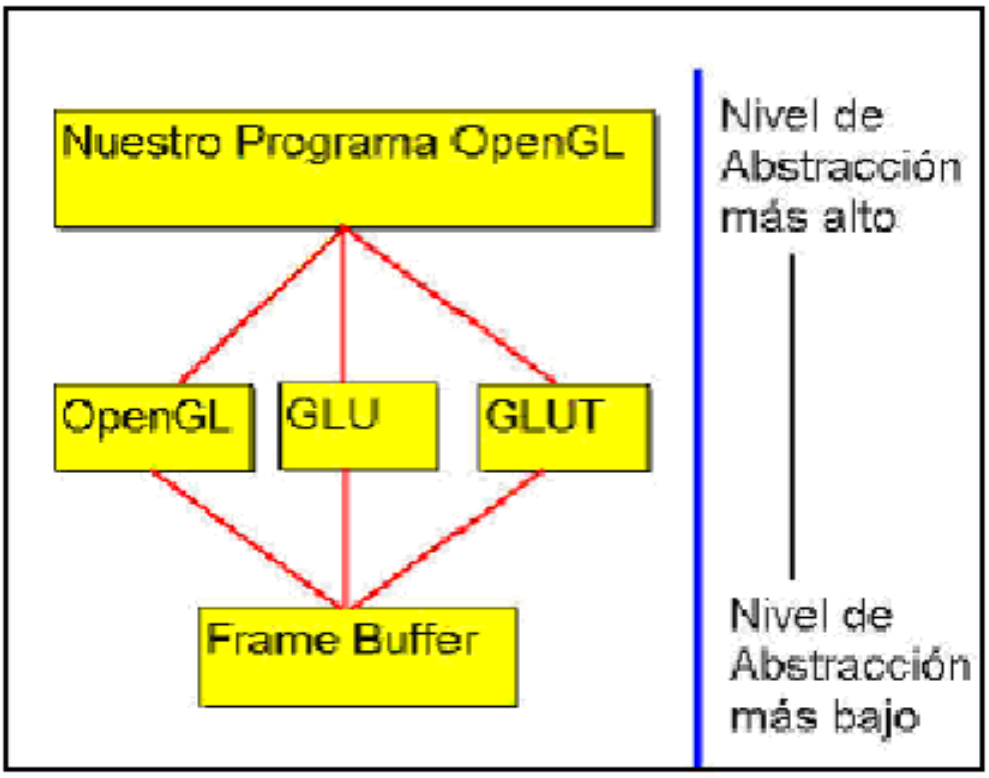
Los desarrolladores de software no necesitan ninguna licencia de OpenGL ni pagar ningun tipo de cuota para usar los controladores OpenGL. OpenGL se admite nativamente en el sistema operativo y los controladores con licencia los proporcionan los propio suministradores de hardware.[1]

1.10 ¿CÓMO FUNCIONA OPENGL?

OpenGL funciona a través de una serie de librerías DLL que suelen tener variados nombres, pero que a fin de cuentas cubren funciones muy específicas y son muy sencillas de identificar, ahora veremos cuales son estas:

OpenGL.DLL (*Open Graphics Library*, *Librería de Gráficos Abierta*).- Esta podemos encontrarla también con el nombre de *OpenGL32.DLL* (para el caso de los sistemas operativos de 32bits) o bien simplemente como *GL.DLL* . Esta es la librería principal, que contiene la mayoría de las funciones que aquí utilizaremos. Las funciones contenidas en esta librería inician con las letras *gl*.*GLU.DLL* (Graphics Utility Library, *Librería de Utilerías de Gráficos*).- También la podemos encontrar como *GLU32.DLL*. Contiene funciones para objetos comunes a dibujar como esferas, donas, cilindros, cubos, en fin, varias figuras ya predefinidas que pueden llegar a ser útiles en ciertos casos, así como funciones para el manejo de la cámara entre muchas otras. Podremos identificar las funciones que pertenecen a esta librería porque llevan antepuestas en su nombre las siglas *glu*. Estas librerías se encuentran disponibles en Internet, y se distribuyen de manera gratuita en diferentes sitios, principalmente se pueden encontrar en el sitio oficial de OpenGL en http://www.opengl.org , donde seguramente se encontrará la de su sistema operativo en específico.

Otro componente es el "*Frame Buffer"*, que no es otra cosa que el área de memoria donde se construyen los gráficos antes de mostrarlos al usuario, es decir, que el programa de OpenGL escribe en esta área de memoria, y automáticamente envía su contenido a la pantalla una vez que la escena está completamente construida. La (FIGURA 3)muestra gráficamente como esta constituida la estructura de abstracción de OpenGL.[2]



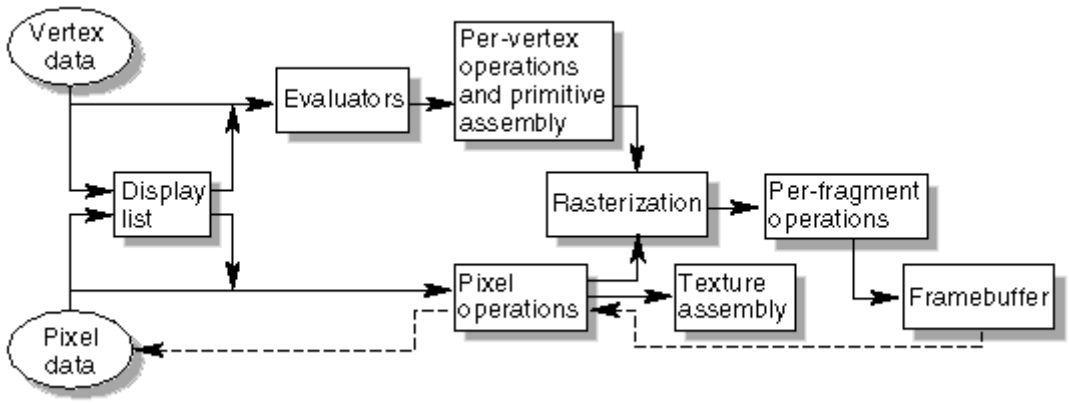
NIVELES DE ABSTRACCION EN OPENGL

(FIGURA 3)

1.11 EL PIPELINE DE RENDERIZADO DE OPENGL

La mayor parte de OpenGL siguen un mismo orden en sus operaciones, una serie de plataformas de proceso, que en su conjunto crean lo que se suele llamar "*OpenGL Rendering Pipeline*".

El siguiente diagrama (FIGURA 4)describe el funcionamiento del pipeline:



(FIGURA 4)

En este diagrama se puede apreciar el orden de operaciones que sigue el pipeline para renderizar. Por un lado esta el “vertex data”, que describe los objetos de la escena, y por el otro lado, el “píxel data ”, que describe las propiedades de la escena que se aplican sobre la imagen tal y como se representa en el buffer. Ambas se pueden guardar en una “display list” que es un conjunto de operaciones que se guardan para ser ejecutadas en cualquier momento.

Sobre el “vertex data” se pueden aplicar “evaluators”, para describir curvas o superficies parametrizadas mediante puntos de control. Luego se aplicaran las “pervertx operations”, que convierten los vértices en primitivas. Aquí es donde se aplican las operaciones geométricas como rotaciones translaciones, etc., por cada vértices.

En la sección de “primittive assembly”, se hace “clipping” de lo que queda fuera del plano de proyección, entre otros.

Por la parte de “píxel data”, están las “píxel operations”. Aquí los píxeles son desempaquetados desde algun array del sistema (como el framebuffer) y tratados (escalados, etc.). luego si estamos tratando con texturas, se preparan en la sección “texture assembly”.

Ambos caminos convergen en la “rasterization”, donde son convertidos en fragmentos Cada fragmento será un píxel del framebuffer . aquí es donde se tiene en cuenta el modelo de sombrado, las lineas, o el antialiassing.
En la ultima etapa, las “per-fragmet operations”, es donde preparan los texeles (elementos de texturas) para se aplicados a cada píxel, la niebla, el z-buffering, el blending, etc. Todas estas operaciones sedembocan en el framebuffer, donde se obtiene el render final.

El funcionamiento esta citado según [2]

1.12 OPENGL COMO UNA MÁQUINA DE ESTADO:

OpenGL es una máquina de estados. Cuando se activan o configuran varios estados de la máquina, sus efectos perdurarán hasta que sean desactivados. Por ejemplo si el color para pintar poligonos se pone en blanco, todos los poligonos se pintarán de esta color hasta cambiar el estado de esa variable.Existen otros estado que funcionan como booleanos (on o off, 0 o 1). Los estados se activan mediante las funciones glEnable y desactivan con glDisable.

Todos los estados tienen un valor por defecto, y tambien alguna función con la que conseguir su valor actual. Estas pueden ser mas generales, del tipo glGetDoublev() o glIsEnabled(), o mas especificas, como glGetLight o glGetError(). Segun [2].

1.13 PERCEPCIÓN 3D

Los gráficos 3D por ordenador son en verdad imágenes bidimensionales en una pantalla plana de ordenador que transmiten una ilusión de profundidad. Para ver realmente una imagen 3D, necesitamos ver el objeto con ambos ojos, o proporcionar a cada uno de ellos imágenes separadas y únicas del objeto. El cerebro combina entonces estas dos imágenes ligeramente diferentes para producir una única imagen 3D compuesta en nuestra cabeza. (FIGURA 5)

(FIGURA 5)

1.14 2DIMENSIONES+PERSPECTIVA=3D

La razón por la que el mundo no se vuelve plano cuando nos cubrimos un ojo es que muchos de los efectos de un mundo 3D están también presentes en un mundo 2D. El indicador más obvio es que los objetos cercanos parecen mayores que los distantes, esto es la perspectiva, que junto con los cambios de color, texturas, iluminación y sombreado alimentan juntos la percepción de una imagen tridimensional. (FIGURA 6)

(FIGURA 6)

1.15 ELIMINACION DE CARAS OCULTAS

La figura anterior tiene la suficiente información para sugerir la apariencia de tres dimensiones, pero no la suficiente para discernir cuál es la cara frontal y cual la trasera. Si el cubo fuera un sólido, no sería posible ver las esquinas posteriores del cubo. Para simular esto en un dibujo bidimensional, las líneas que estarían oscurecidas por superficies frente a ellas deben eliminarse. (FIGURA 7)

(FIGURA 7)

1.16 COLORES Y SOMBREADO

La figura anterior todavía no se parece mucho a un objeto real, que tendría algo de color y textura. Si todo lo que hacemos es colorear el dibujo y dibujarlo en 2D se obtiene algo como la (FIGURA 8).

(FIGURA 8)

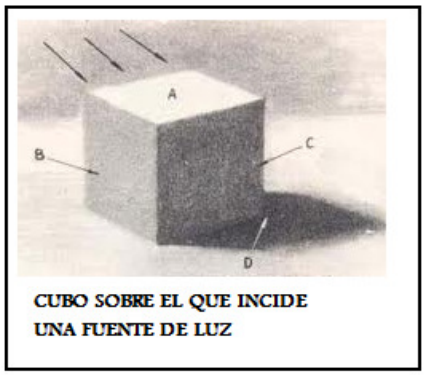
De nuevo el objeto parece bidimensional. Para recuperar la perspectiva de un objeto sólido, se necesita hacer cada una de las tres caras de un color diferente, o hacerlas del mismo color con sombreado para producir la ilusión de iluminación. (FIGURA 9)

CUBO CON COLOR

(FIGURA 9)

1.17 LUCES Y SOMBRAS

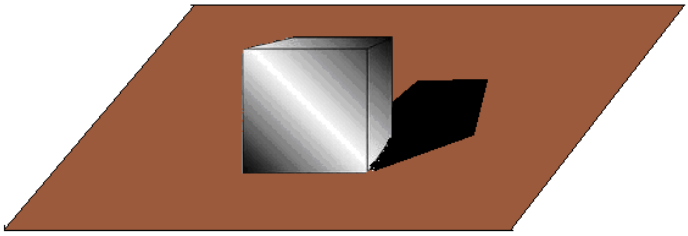
La iluminación tiene dos efectos importantes sobre los objetos vistos en tres dimensiones. Primero causa que una superficie de un color uniforme aparezca sombreada cuando se ilumina desde un ángulo. Segundo, los objetos que no transmiten luz arrojan una sombra cuando obstruyen el camino de un rayo. (FIGURA 10)



CUBO SOBRE EL QUE INCIDE
UNA FUENTE DE LUZ

(FIGURA 10)

Dos fuentes de luz pueden influir en nuestros objetos tridimensionales. La luz ambiente, que es indirecta, es una iluminación uniforme que puede causar efectos de sombreado en objetos de un color sólido; la luz ambiente hace que las aristas distantes aparezcan oscuras. Otra fuente de luz se denomina *lámpara*, y se puede usar para cambiar el sombreado de un objeto sólido y para efectos de sombra. (FIGURA 11)

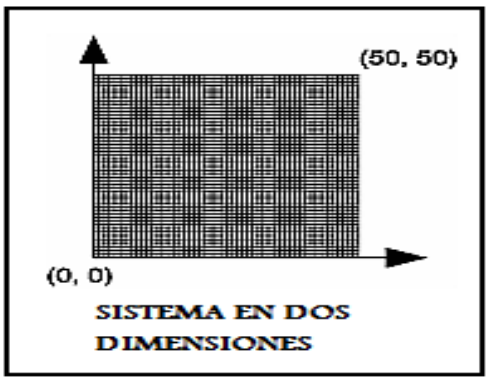


CUBO CON SOMBRA PARA AUMENTAR REALISMO

(FIGURA 11)

1.18 SISTEMAS DE COORDENADAS

Ahora que ya sabemos cómo puede percibir el ojo tres dimensiones en una superficie bidimensional, consideremos cómo dibujar los objetos en la pantalla. En OpenGL, cuando creamos una ventana para dibujar en ella, hay que especificar el sistema de coordenadas que deseamos usar, y cómo mapear las coordenadas específicas sobre los pixels físicos de la pantalla.(FIGURA 12)

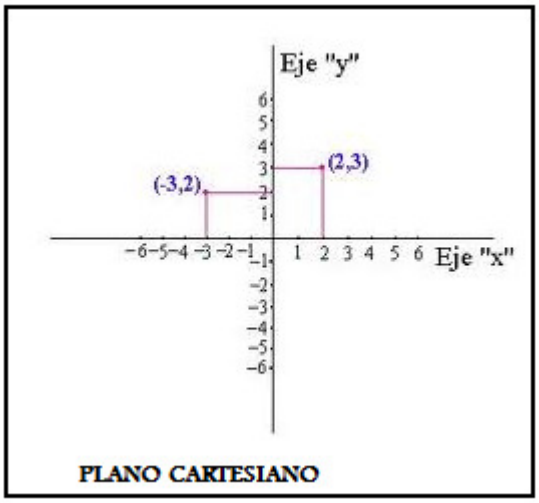


(FIGURA 12)

1.19 COORDENADAS CARTESIANAS 2D

El sistema de coordenadas más común para el dibujo bidimenional es el sistema *Cartesiano*. Las coordenadas cartesianas se especifican por una coordenada x y una coordenada y. La coordenada x es un medida de posición en la dirección horizontal, y la coordenada y es una medida de posición en la dirección vertical.

El origen de coordenadas se encuentra en x=0, y=0. Las coordenadas se describen como pares entre paréntesis, separados por una coma: (0,0). (FIGURA 13)

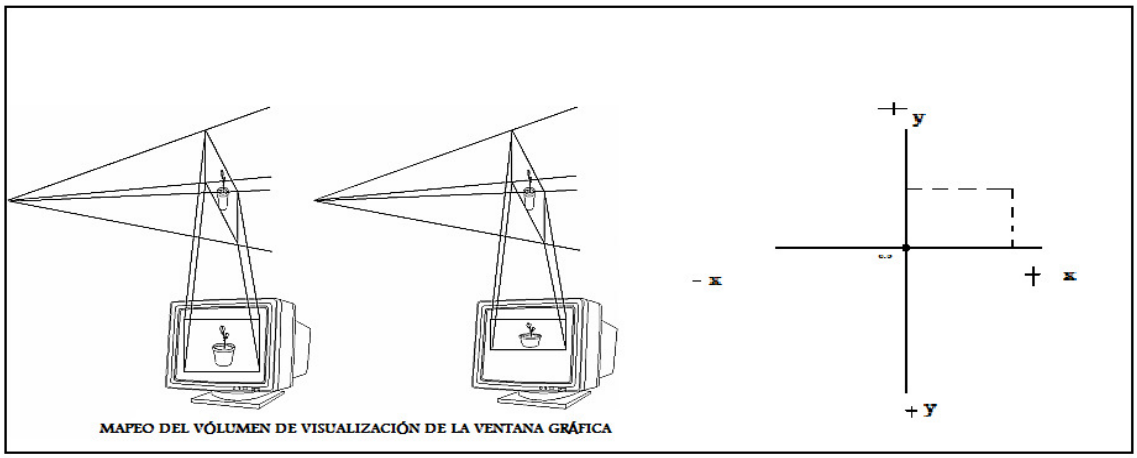


(FIGURA 13)

1.20 LOCALIZACION DE LAS COORDENADAS DE TRABAJO

Una ventana se mide físicamente en términos de pixels, debe decirle a OpenGL cómo trasladar los pares de coordenadas especificados a coordenadas de la ventana.

Esto se hace especificando la región del espacio cartesiano que ocupa la ventana; este espacio se conoce como el *área de trabajo*. En el espacio bidimensional, el área de trabajo es el máximo y mínimo valor x e y que hay dentro de la ventana. Usando las funciones de OpenGL, es posible volver el sistema de coordenadas de arriba a abajo o de derecha a izquierda. De hecho, el mapeado por defecto de las ventanas de windows tiene los valores positivos de las y descendiendo de arriba a abajo, lo que puede ser molesto para dibujar gráficos. (FIGURA 14)



(FIGURA 14)

1.21 VISTAS

Raramente coincidirán el alto y el ancho de su área de trabajo con el alto y ancho de la ventana en pixels. Por tanto, hay que mapear las coordenadas cartesianas lógicas sobre las coordenadas físicas en pixels de la pantalla. Este mapeado está especificado por un montaje conocido como *vista*. La vista es la región dentro del área cliente de la ventana que se usará para dibujar dentro del área de trabajo. La vista simplemente mapea el área de trabajo en una región de la ventana. Normalmente se define como la ventana entera, pero no es estrictamente necesario.

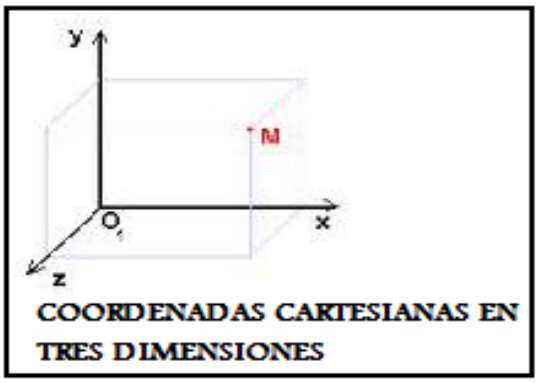
Podemos usar las vistas para encoger o agrandar la imagen dentro de la ventana y para mostrar sólo una porción del área de trabajo definiendo una vista más grande que el área cliente de la ventana.

1.22 EL VÉRTICE: PRIMITIVAS DE DIBUJO

Tanto en 2D como en 3D, cuando se dibuja un objeto lo que se hace es componerlo con muchas formas más pequeñas denominadas *primitivas*, que son superficies de una o dos dimensiones como puntos, líneas y polígonos (una forma de múltiples lados plana) que se ensamblan en el espacio para crear objetos 3D. Por ejemplo, un cubo tridimensional , está hecho con seis cuadrados bidimensionales, colocado cada uno en una cara diferente. Cada esquina del cuadrado, o de cualquier primitiva, se llama *vértice*. Estos vértices se definen para ocupar unas coordenadas particulares en el espacio.

1.23 COORDENADAS CARTESIANAS 3D

Ahora se extenderá el sistema de coordenadas bidimensional a la tercera dimensión y se añadirá un componente de profundidad. La siguiente figura muestra el sistema de coordenadas cartesiano con un nuevo eje, el z, que es perpendicular tanto al x como al y. Ahora se especifica un una posición en el espacio mediante tres coordenadas, x,y,z. (FIGURA 15)



(FIGURA 15)

1.24 LA ESENCIA DE LAS 3D

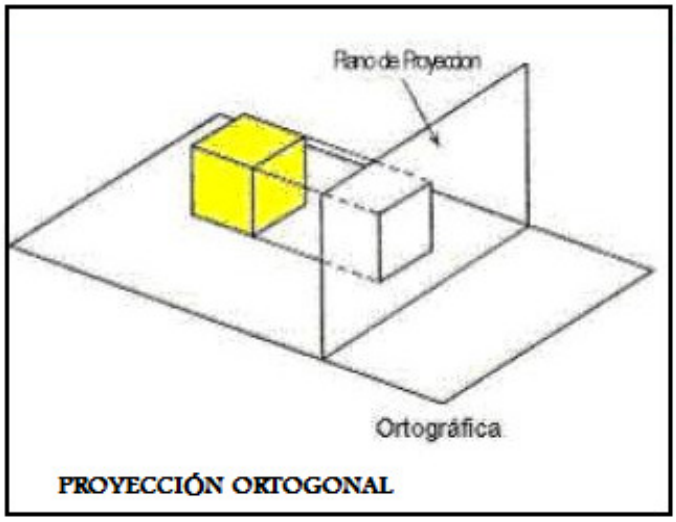
Se ha visto cómo especificar una posición 3D mediante el uso de coordenadas cartesianas sin importar de qué manera se engañe al ojo, sin embargo los píxeles de la pantalla sólo tienen dos coordenadas. OpenGL transforma las coordenadas 3D a coordenadas tridimensionales mediante trigonometría y manipulación de matrices, aunque ahora no es momento de entrar en el tema.

Lo que necesita entender es el concepto de *proyección*. Las coordenadas 3D se proyectan en una superficie 2D (el fondo de la ventana). Al especificar la proyección se especifica el volumen de trabajo que se desea visualizar en la ventana y cómo debe trasladarse.

1.25 PROYECCIONES ORTOGRAFICAS

Definimos esta proyección si especificamos un espacio de trabajo rectangular. Cualquier cosa que esté fuera de este área no se dibuja. Además todos los objetos que tienen las mismas dimensiones aparecen con el mismo tamaño, sin tener en cuenta si están lejos o cerca.

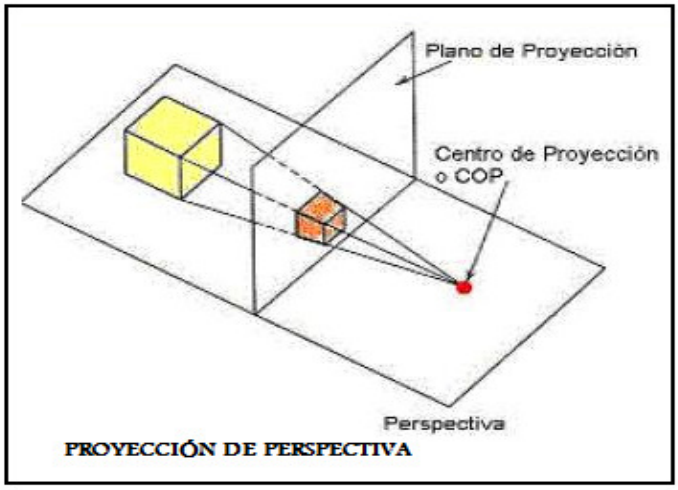
En una proyección ortográfica especificamos el volumen de trabajo al especificar los planos de trabajo de lejos, cerca, izquierda, derecha, arriba y abajo. Los objetos que coloquemos dentro de este espacio de visualización se proyectan en una imagen 2D que aparece en la pantalla.(FIGURA 16)



(FIGURA 16)

1.26 PROYECCIONES EN PERSPECTIVA

Esta proyección añade el efecto de que los objetos más distantes aparecen más pequeños que los cercanos. El volumen de visualización se parece a una pirámide con el pico cortado. Los objetos más cercanos a la parte frontal del volumen aparecen casi con su tamaño real, mientras que los objetos cercanos a la parte trasera del volumen se encogen al proyectarse sobre la parte frontal. Este tipo de proyección da el mayor realismo para simulación y animación 3D. (FIGURA 17)



(FIGURA 17)

CAPÍTULO 2

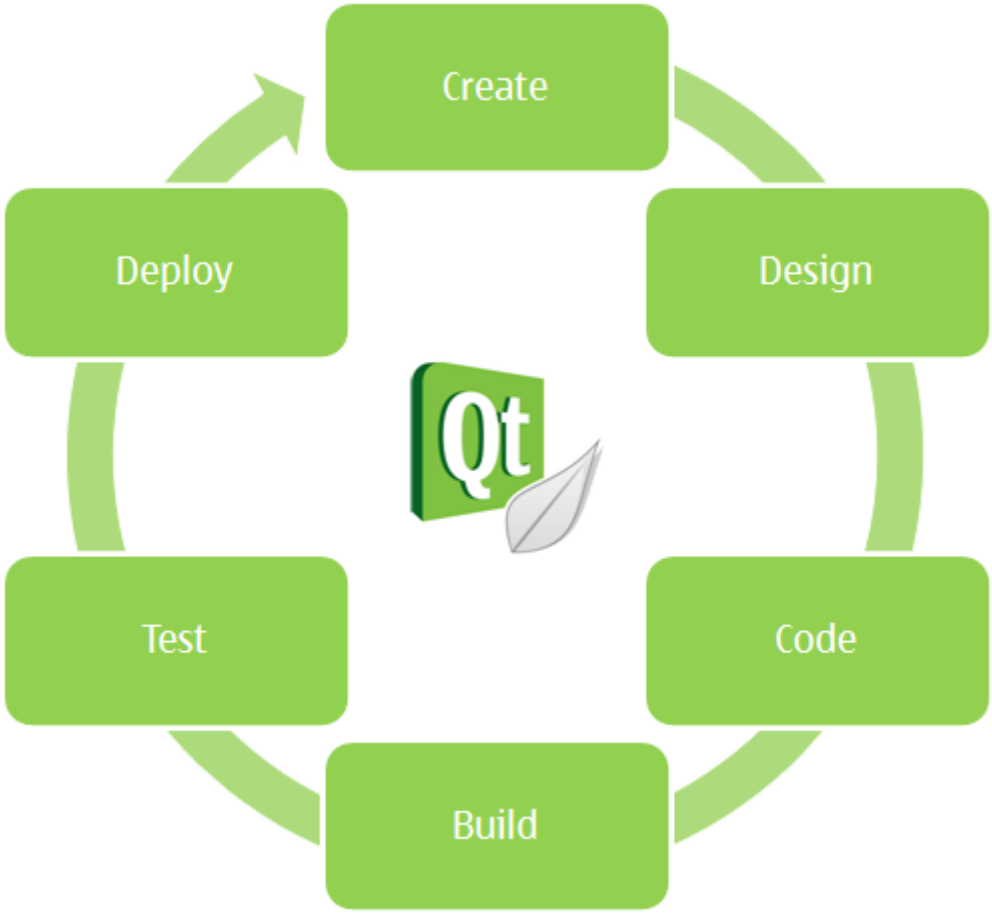
QTCREATOR

En este capítulo se hará una introducción a los conceptos básicos de QtCreator, y se dará la información básica y necesaria para poder desenrrollarse de manera adecuada en el escritorio del programa.

- **Interfaz de usuario**
- **Cambiar el idioma**
- **Navegando por el contenido del proyecto**
- **Visualización de archivos de proyecto**
- **Viendo el sistema de archivos**
- **Visualización de la jerarquía de las clases**
- **Viendo elementos QML**
- **Viendo jerarquía de tipos**
- **Ver la salida**

2.1 Que es QtCreator?

Qt Creator es un entorno de desarrollo integrado (IDE) que le proporciona las herramientas necesarias para diseñar y desarrollar aplicaciones con el framework de aplicaciones Qt. Está diseñado para desarrollar aplicaciones e interfaces de usuario y desplegarlas a través del escritorio y sistemas operativos móviles. Proporciona herramientas para llevar a cabo todo el desarrollo de aplicaciones, desde la creación de un proyecto hasta el despliegue de la aplicación en las plataformas de destino.**(FIGURA 18)**



(FIGURA 18)

2.2 El desarrollo multiplataforma

Una de las principales ventajas de Qt Creator es que permite que un equipo de desarrolladores puedan compartir un proyecto a través de diferentes plataformas de desarrollo con una herramienta común para el desarrollo y depuración.

2.3 Definiciones básicas para enfrentarse a Qt creator

En programación, un widget (o control) es un elemento de una interfaz (interfaz gráfica de usuario o GUI) que muestra información con la cual el usuario puede interactuar. Por ejemplo: ventanas, cajas de texto, checkboxs, listbox, entre otros.

En otras palabras, los widgets son bloques básicos y visuales de construcción que, combinados en una aplicación, permiten controlar los datos y la interacción con los mismos.

No debe confundirse con los widgets de escritorio, que son pequeñas aplicaciones que proveen información visual al usuario y que permite ser fácilmente accedida: relojes, calendarios, calculadoras, notas y demás programas de escritorio.

Durante la ejecución de una aplicación un widget puede estar activado o desactivado. El widget activo responde a eventos (presionar una tecla, acciones del mouse, etc). En tanto el widget inactivo no responde a ningún evento y suele tener una apariencia diferente del activado (colores menos llamativos).[7]

2.4 Widget tookits:

Existen los widget tookits, son paquetes de widgets genéricos que permiten a los programadores desarrollar aplicaciones gráficas. En general, cada tipo de widget es definido como una clase en la programación orientada a objetos; luego, a partir de la herencia de clase, se crean muchos widgets derivados.

La programación orientado a objetos no es mas que la representación detallada y particular de algo de la realidad. Todo objeto tiene un identidad o nombre, estado (características definidas generalmente en variables) y comportamiento (sus funciones o procedimientos).

Una forma de alterar el estado de un objeto es a través de sus funciones.

Las clases son generalizaciones de un objeto en particular. Por ejemplo, el objeto Auto pertenece a la clase Autos. Una instancia de una clase es siempre un objeto único.

En programación, mecanismo que permite derivar características de una clase a otra y así extender sus funcionalidades. Una de sus funciones más importantes es proveer polimorfismo. Existen dos tipos de herencias:

> Herencia simple: una clase sólo puede heredar características de una sola clase, o sea, puede tener un padre. Smalltalk, Java y Ada soportan herencia simple.[7]

Herencia múltiple: una clase puede heredar características de una o más clases, por lo tanto, puede tener varios padres. C++ soporta herencia múltiple.
La herencia es una de las características de los lenguajes del paradigma orientado a objetos.

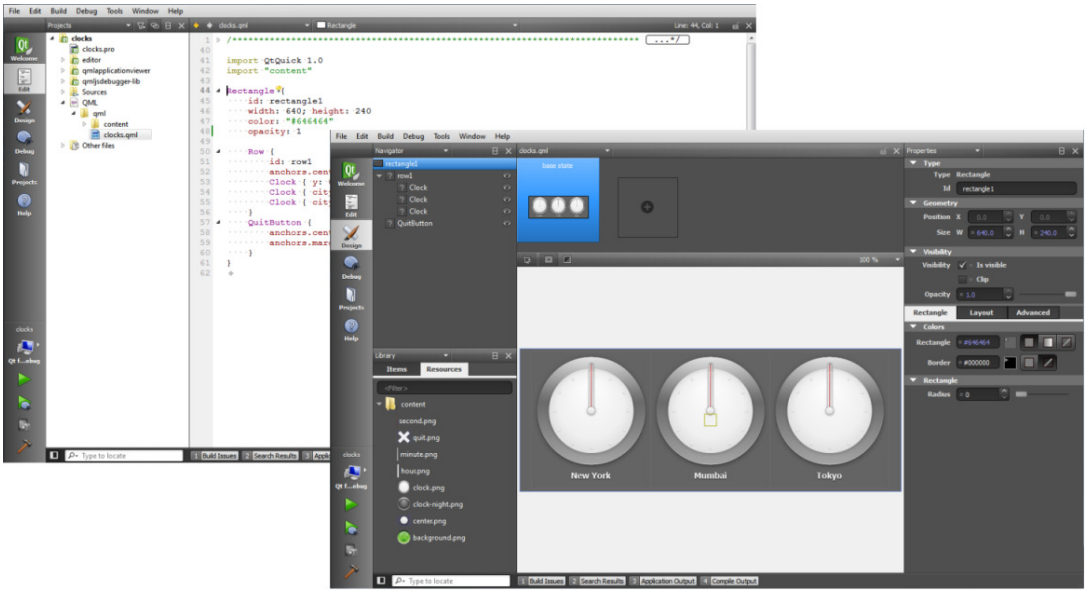
2.5 Interfaz de usuario:

(FIGURA 19)

Al iniciar Qt creator(FIGURA 19) , se abre el modo de **bienvenida,** donde se puede:

- Leer noticias de los laboratorios de Qt
- Tutoriales y ejemplos de proyectos abiertos
- Crear y abrir proyectos
- Envíenos sus comentarios al equipo de desarrollo
- Abrir las últimas sesiones y los proyectos

Se puede utilizar el selector de modo para cambiar a otro modo de Qt Creator. La siguiente imagen (FIGURA 20)muestra un ejemplo de aplicación en modo de edición y el modo de diseño, aunque para este proyecto de debe usar el modo edición para poder integrar la biblioteca opengl en Qt creator.



(FIGURA 20)

2.6 Cambiar el idioma:

Qt Creator ha sido traducido a varios idiomas. Para cambiar el idioma, seleccionar **Herramientas> Opciones> Entorno** y seleccione el idioma en el campo **del lenguaje.** El cambio tendrá efecto después de reiniciar Qt Creator.

Modos:

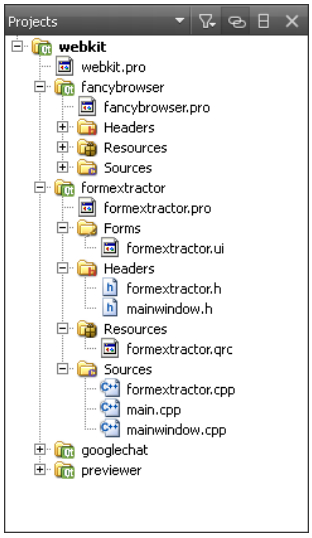
El selector de modo permite cambiar rápidamente entre tareas como la edición de archivos de proyecto y la fuente, el diseño de interfaces de aplicación, la configuración de cómo los proyectos son construidos y ejecutados, y la depuración de aplicaciones. Para cambiar los modos, haga clic en los iconos.

Puede usar Qt Creator en las siguientes modalidades:

- **Bienvenido** modo para los proyectos de apertura.
- **Editar** el modo de edición de proyectos y archivos de código fuente.
- **Diseño de** modo de diseñar y desarrollar interfaces de usuario. Este modo está disponible para los archivos de la interfaz de usuario (Ui o QML).
- **Depurar** el modo de inspeccionar el estado de su programa durante la depuración.
- **Proyectos** para configurar el modo de construcción y ejecución del proyecto. Este modo está disponible cuando hay un proyecto abierto.
- **Analizar** el modo para utilizar las herramientas de análisis de código para detectar pérdidas de memoria y el perfil de C + + o el código QML.
- **Ayudar a** modo de visualizar la documentación de Qt.

Ciertas acciones en desencadenan un cambio de modo. Al hacer clic en **Depurar> Iniciar depuración> Iniciar depuración** cambia automáticamente al modo de **depuración.**

2.7 Navegando por el contenido del proyecto:



La barra lateral está disponible en los modos **de edición** y **depuración.** Utilice la barra lateral para navegar por los proyectos, archivos y favoritos, y para ver la jerarquía de clases.(FIGURA 21)

(FIGURA 21)

Puede seleccionar el contenido de la barra lateral en el menú lateral:

- **Proyectos** muestra una lista de proyectos abiertos en la sesión actual.
- **Los documentos abiertos** en la actualidad muestra los archivos abiertos.
- **Marcadores** muestra todos los marcadores de la sesión actual.
- **Sistema de archivos** muestra todos los archivos en el directorio seleccionado.
- **Vista de clases** muestra la jerarquía de clases de los proyectos abiertos.
- **Esquema** muestra la jerarquía de símbolos de un archivo de C + + y la jerarquía de elementos de un archivo QML.
- **Jerarquía de tipos** muestra las clases de base de una clase.

Puede cambiar la vista de la barra lateral de la siguiente manera:

- Para dividir la barra lateral, haga clic en  . Seleccione el nuevo contenido para ver en la vista dividida.
- Para cerrar una vista lateral, haga clic en  .

Las opciones adicionales en cada vista se describen en las secciones siguientes.

2.8 Visualización de archivos de proyecto:

La barra lateral muestra los proyectos en un árbol del proyecto. El árbol del proyecto contiene una lista de todos los proyectos abiertos en la sesión actual. Los archivos de cada proyecto se agrupan de acuerdo a su tipo de archivo.

Usted puede utilizar el árbol del proyecto de la siguiente manera:

- Para que aparezca un menú contextual que contiene las acciones más necesarias botón derecho del ratón en un elemento del árbol del proyecto. Por ejemplo, a través del menú del directorio raíz del proyecto puede, entre otras acciones, construir, reconstruir, limpiar y ejecutar el proyecto.
- Para ocultar las categorías y ordenar los archivos del proyecto en orden alfabético, haga clic en **Simplificar** y seleccione **Árbol.**
- Para ocultar los archivos de origen que son generados automáticamente por el sistema de construcción, durante una generación, haga clic en y seleccione **Ocultar archivos generados.**
- Para mantener la posición en el árbol del proyecto sincronizado con el archivo abierto en el editor, haga clic en .

Para ver la ruta absoluta a un archivo, mueva el puntero del ratón sobre el nombre del archivo.

2.8.1 Viendo el sistema de archivos:

Si no puede ver un archivo en la vista de **los proyectos,** cambiar a la vista **el sistema de archivos,** que muestra todos los archivos en el sistema de archivos.

Para mantener la posición en el árbol sincronizado con el archivo abierto en el editor, haga clic en .

2.8.2 Visualización de la jerarquía de las clases:

La **Vista de clases** muestra la jerarquía de clases de los proyectos abiertos. Para ordenar la lista por subproyectos, haga clic en .

2.8.3 Viendo elementos QML:

La vista de **esquema** muestra la jerarquía de elementos en un archivo QML.

- Para ver una lista completa de todos los enlaces, haga clic en y seleccione **Mostrar todos los enlaces.**
- Para mantener la posición en la vista sincronizada con el elemento seleccionado en el editor, haga clic en .

2.8.4Viendo jerarquía de tipos:

Para ver las clases base de una clase, haga clic en la clase y seleccione **Jerarquía de tipo abierto** o presione **Ctrl + Shift + T.**

2.9 Ver la salida:

El panel de tareas puede mostrar uno de los siguientes paneles:

- **Crear problemas**
- **Resultados de la búsqueda**
- **Solicitud de salida**
- **Compilación de salida**
- **Mensajes generales**
- **Control de versiones**

Paneles de resultados están disponibles en todos los modos . Haga clic en el nombre de un panel de salida para abrir el panel. Para maximizar un panel de salida abierta, haga clic en el botón **de salida Maximizar panel** o pulse **Alt +9.**

> Para buscar dentro de la **producción de aplicaciones** y **compilar** los paneles **de salida,** pulse **Ctrl + F** cuando el panel está activa. Introducir criterios de búsqueda en el campo

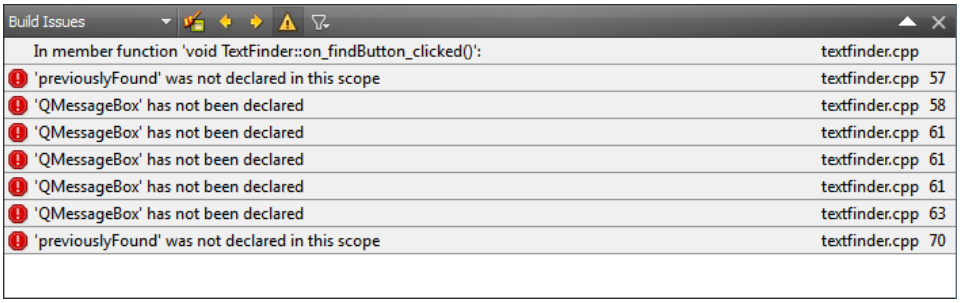
Buscar y haga clic en las flechas izquierda y derecha para buscar hacia arriba y abajo en el panel.

Para abrir los **mensajes generales** y los paneles de **control de versiones,** seleccione **Ventana> Paneles de salida.**

2.10 PANEL DE LISTA DE ERRORES:

El panel **lista de Problemas** proporciona una lista de errores y avisos encontrados durante una compilación. Los filtros de panel de salida irrelevante desde las herramientas de construcción y presenta los temas de una manera organizada.

Al hacer clic derecho en una línea, aparecerá un menú contextual con opciones para copiar el contenido y mostrar un control de versiones vista de anotación de la línea que hace que el mensaje de error. (FIGURA 22)

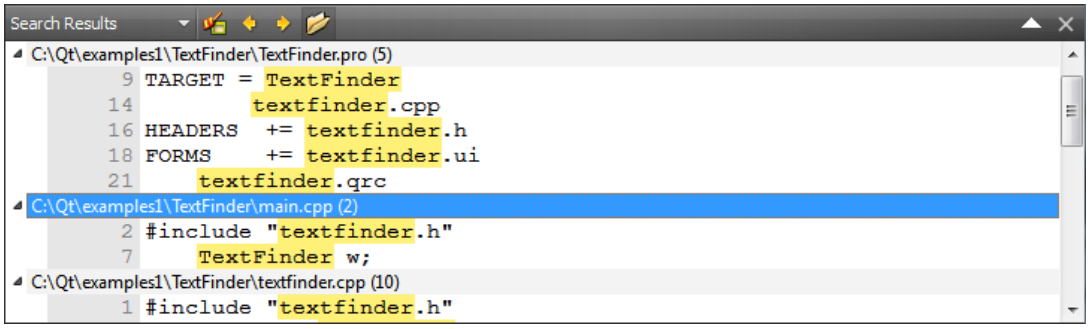


(FIGURA 22)

2.11 Resultados de la búsqueda:

El panel **de resultados de búsqueda** muestra los resultados de las búsquedas globales, por ejemplo, buscar dentro de un documento, los archivos en el disco, o todos los proyectos.

La siguiente figura (FIGURA 23)muestra un resultado de búsqueda de ejemplo para todas las apariciones de `textfinder` dentro de la `"/ TextFinder"` directorio.

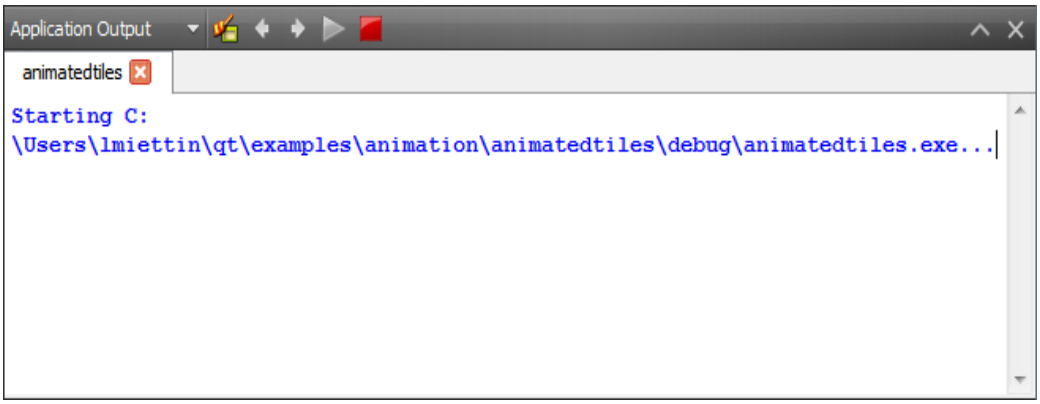


(FIGURA 23)

2.12 Solicitud de salida:

El **panel** de resultados **de la aplicación** muestra el estado de un programa cuando se ejecuta, y el resultado de la depuración.

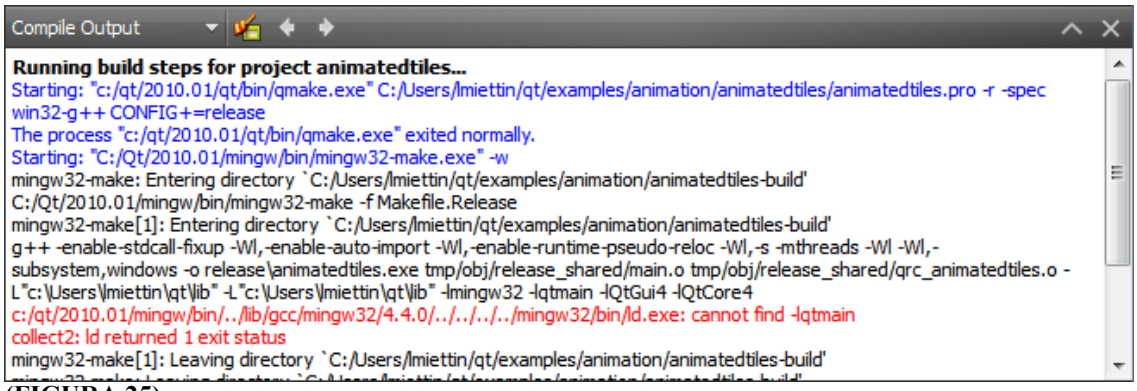
La siguiente figura(FIGURA 24) muestra un ejemplo del resultado de qDebug().



(FIGURA 24)

2.13 Compilación de salida :

El panel **de salida de compilación** proporciona toda la salida del compilador. La **salida de compilación** es una versión más detallada de la información que aparece en el panel **lista de errores.**(FIGURA 25)



(FIGURA 25)

CAPÍTULO 3

ASPECTOS GENERALES DE CÓMO CREAR UN PROYECTO EN CON QTCREATOR

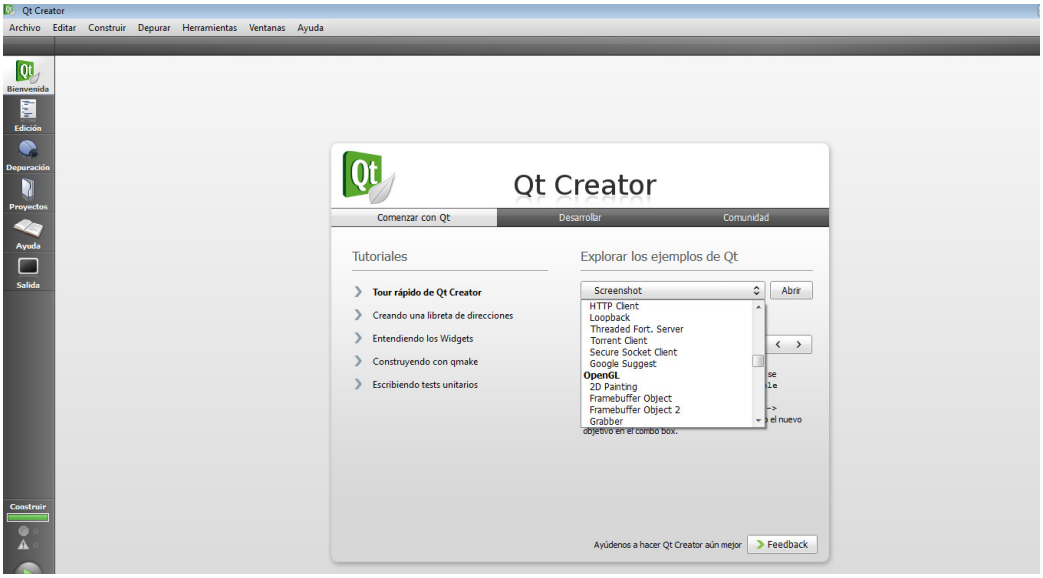
En este capitulo se mostrarán las herramientas que ofrece el programa para poder desarrollar mejor la aplicación .

- Creación y ejecución de una aplicación de ejemplo
- Gestión de proyectos:
- Crear un proyecto
- Cómo añadir archivos a los proyectos
- Creación de clases C + +
- Creación de fragmentos de OpenGL y Vertex Shaders
- Integración
- Viendo otros tipos de archivo en el panel de Proyectos
- Añadir subproyectos a proyectos
- Apertura de un proyecto
- Añadir Bibliotecas a los proyectos
- Ejemplo de Adición de bibliotecas internas
- Gestión de sesiones

3.0 Creación y ejecución de una aplicación de ejemplo:

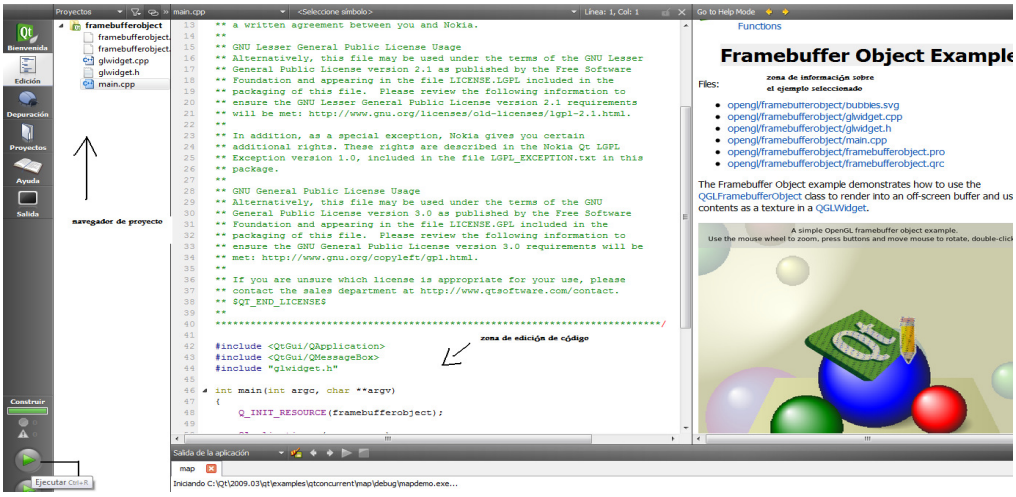
Para comprobar que la instalación es correcta se puede hacer mediante la apertura de un proyecto de ejemplo de aplicación existente.

1. En la página **Bienvenida,** seleccione la casilla **explorar los ejemplos de Qt** a continuación, busque Framebuffer Object **Example** en la lista de ejemplos. (FIGURA 26)



(FIGURA 26)

2. En la pantalla que le aparecerá puede verse las distintas zonas en que se divide por defecto. Hay que dar un clic en el símbolo de "ejecutar" para que compile y ejecute el programa.(FIGURA 27)



(FIGURA 27)

3. Sí todo esta bien el resultado debe de ser el siguiente(FIGURA 28)

(FIGURA 28)

De esta se asegura que todo esta correcto además de corroborar que la librería de opengl esta integrada de forma correcta en Qt creator.

3.1 Gestión de proyectos:

Para configurar un proyecto, primero tiene que decidir qué tipo de aplicación se quiere desarrollar: Si desea una interfaz de usuario basada en Qt Quick, widgets Qt, o HTML5.

Para un proyecto de Qt Quick o HTML5, también debe elegir el idioma para implementar la lógica de la aplicación: C + + o JavaScript . También se pueden crear otros tipos de proyectos, tales como Qt aplicaciones de consola, compartida o estática C + + bibliotecas, o subproyectos.

Se puede utilizar asistentes para crear e importar proyectos. Los asistentes le pedirá que introduzca los ajustes necesarios para el tipo de proyecto y pueda crear los archivos.

Se puede agregar sus propios asistentes personalizados para estandarizar la forma en subproyectos y las clases que se agrega a un proyecto.

Se trata de un sistema multi-plataforma para construir la automatización que ayuda a simplificar el proceso de construcción para proyectos de desarrollo en diferentes plataformas. Qmake automatiza la generación de configuraciones de construcción de manera que con sólo unas pocas líneas de información se pueda crear cada configuración.

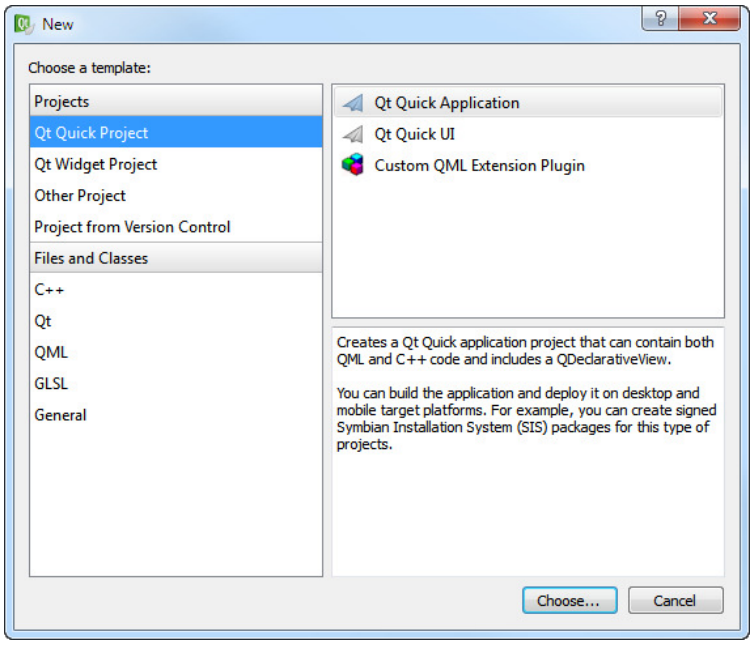
Puede modificar la configuración de crear y ejecutar proyectos de qmake en el modo de **Proyectos.**

Alternativamente, se puede utilizar el sistema de automatización de generación CMake y establecer los proyectos de forma manual. Además, se puede importar proyectos genéricos que no utilizan qmake o CMake. Esto permite usar Qt Creator como un editor de código. Para los proyectos genéricos, Qt Creator hace caso omiso de su sistema de construcción.

Puede utilizar sesiones para almacenar datos de carácter personal, tales como los marcadores y puntos de interrupción que no suelen ser de interés para los desarrolladores que trabajan en los mismos proyectos. Las Sesiones permiten cambiar rápidamente entre los proyectos cuando se trabaja en varios proyectos.

3.2 Crear un proyecto

Para crear un proyecto nuevo, seleccionar **Archivo> Nuevo o proyecto** y seleccionar el tipo de proyecto. El contenido de los diálogos del asistente dependerá del tipo de proyecto y los objetivos de construcción que se selecciona en el cuadro de diálogo **Configuración de destino.** Siga las instrucciones del asistente.(FIGURA 29)



(FIGURA 29)

Puede utilizar asistentes para crear los siguientes tipos de proyectos:

- Qt Quick Proyecto

  Utilizar elementos o componentes QML Qt Quick para definir la interfaz de usuario y, opcionalmente, C + + o JavaScript para definir la lógica de la aplicación.

- Qt Widget Proyecto

  Utilice formas *Qt Designer* para definir una interfaz de usuario basada en Qt flash y C + + para definir la lógica de la aplicación

- Otro proyecto
  - Aplicaciones basadas en HTML5
  - Qt aplicaciones de consola
  - Compartida o estática C + + bibliotecas
  - Qt pruebas unitarias
  - Qt diseño personalizado Widgets
  - Subproyectos
- Proyecto de control de versiones

  Importar un proyecto de un sistema de control de versiones compatibles.

Para cambiar la ubicación del directorio del proyecto, y para especificar la configuración para crear y ejecutar proyectos, seleccione **Herramientas> Opciones ...> Proyectos> General.**

Para especificar construir y ejecutar la configuración de las plataformas de destino diferentes, selección de **proyectos.**

3.3 Cómo añadir archivos a los proyectos :

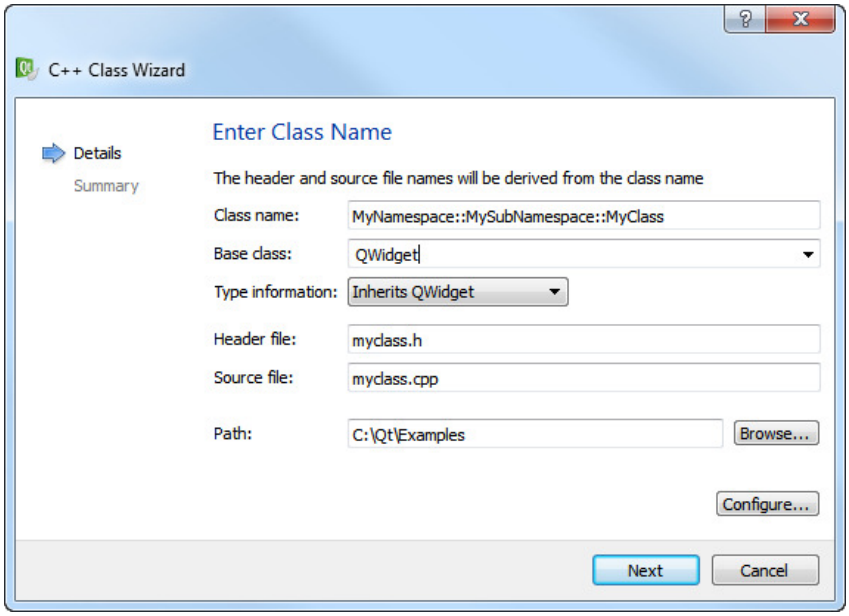
Puede utilizar el asistente también para agregar archivos individuales para sus proyectos. Puede crear los siguientes tipos de archivos:

- Qt archivos de recursos, que le permiten almacenar archivos binarios en el ejecutable de la aplicación
- Formas y clases de *Qt Designer Qt Designer* forma, que especifique las partes de las interfaces de usuario Qt en proyectos basados en flash
- QML archivos, que especifican los elementos de Qt proyectos rápida
- GLSL archivos que definen el fragmento y vertex shaders, tanto en proyectos de Qt Quick y Qt proyectos basados en flash
- Clase C + +, código fuente, o archivos de cabecera que se pueden utilizar para escribir la lógica de la aplicación tanto en proyectos de Qt Quick y Qt proyectos basados en flash
- JavaScript archivos que se pueden utilizar para escribir la lógica de aplicaciones en Qt proyectos rápida
- Los archivos de texto

3.4 Creación de clases C + +

El **C + + Asistente de clases** le permite crear una cabecera de C + + y archivo de origen de una nueva clase que se pueden agregar a un proyecto de C + +. Especificar el nombre de la clase, la clase base, y archivos de cabecera y fuente de la clase.

El asistente admite espacios de nombres. Para utilizar un espacio de nombres, escriba un nombre de clase calificado en el campo **Nombre de clase.** Por ejemplo: MyNamespace:: MySubNamespace:: MyClass. (FIGURA 30)



(FIGURA 30)

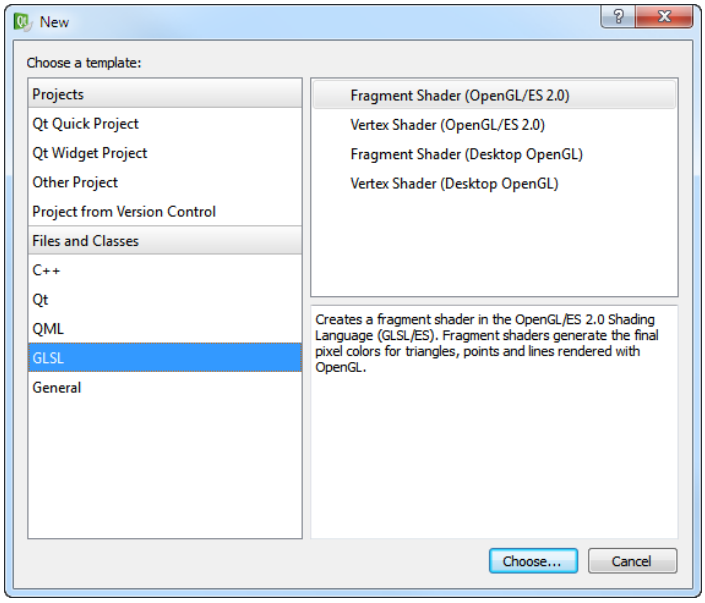
Los nombres de los archivos de cabecera y la fuente se basan en el nombre de la clase. Para cambiar el sufijo por defecto de un archivo, haga clic en **Configurar.**

3.5 Creación de fragmentos de OpenGL y Vertex Shaders :

Qt proporciona apoyo a la integración con las implementaciones de OpenGL en todas las plataformas, lo que permite mostrar el hardware de gráficos 3D acelerados junto con una interfaz de usuario más convencional.

Puede utilizar el QGLShader clase para compilar shaders OpenGL escrito en el OpenGL Shading Language (GLSL) y en el lenguaje de sombreado OpenGL / ES (GLSL / ES). QGLShader y QGLShaderProgram refugio de los detalles de la compilación y la vinculación de vertex y pixel shaders.

Puede usar Qt Creador editor de código para escribir fragmentos y vertex shaders en GLSL o GLSL / ES. El editor de código proporciona resaltado de sintaxis y autocompletado de código para los archivos. (FIGURA 31)



(FIGURA 31)

3.6 Viendo otros tipos de archivo en el panel de Proyectos :

Qt Creator determina si se debe mostrar los archivos de la carpeta del proyecto en el panel de **proyectos** en función del tipo de archivo (. Pro,. PRI. Cpp,. H,. Ui,. PRC, y así sucesivamente). Para mostrar otros tipos de archivos, editar el archivo de proyecto. Añadir los nombres de archivo como valores de la variable `OTHER_FILES`. También puede usar comodines.

Por ejemplo, el siguiente código especifica que los archivos de texto se muestran en el panel de **Proyectos:**

```
OTHER_FILES + = *. Txt
```

Esto también hace que los archivos sean disponibles en el **localizador.**

3.7 Añadir subproyectos a proyectos :

Cuando se crea un nuevo proyecto, puede agregarlo a otro proyecto como un subproyecto en el diálogo de **gestión de proyectos.** Sin embargo, la raíz del proyecto debe especificar que qmake `subdirectorios` utiliza la plantilla para construir el proyecto.

Para crear un proyecto de raíz, seleccione **Archivo> Nuevo o proyecto ... Proyecto> Otros> subdirs Proyecto> Seleccionar.**

En la página **Resumen,** pulse **Finalizar y Agregar subproyecto** para crear el proyecto de raíz y para agregar otro proyecto, como una biblioteca de C + +.

El asistente crea un archivo de proyecto (. Pro) que define una plantilla de `subdirectorios` y el subproyecto que se agrega como un valor de la variable de SUBDIRS. También agrega todos los archivos necesarios para el subproyecto.

Para añadir más subproyectos, haga clic en el nombre del proyecto en el panel **Proyectos** y seleccione **Nuevo Subproyecto** en el menú contextual.

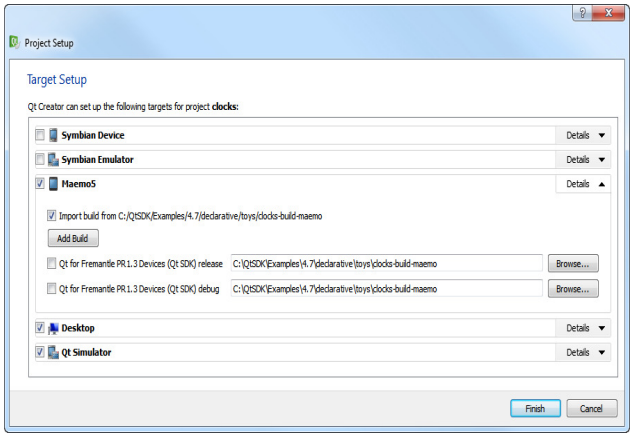
Para eliminar los subproyectos, haga clic en el nombre del proyecto en el panel de **proyectos** y seleccione **Quitar del subproyecto** en el menú contextual.

Para especificar las dependencias, utilice el asistente **Add Library.**

3.8 Apertura de un proyecto

Si Qt Creator no puede encontrar el archivo cuando se abre un proyecto existente, se le pedirá que introduzca la información. Si ha creado el proyecto mediante otra instancia Creador Qt, Qt Creator le pregunta si desea usar la configuración anterior. Los ajustes son específicos para el entorno de desarrollo, y no debe ser copiada de un entorno a otro. Por lo tanto, le recomendamos que haga clic en **No** e ingrese la información de nuevo en el cuadro de diálogo **Configuración de proyectos.**

El cuadro de diálogo **Configuración del proyecto** muestra una lista de entornos de desarrollo para plataformas de destino (por ejemplo, de escritorio, dispositivos Maemo5, y los dispositivos Symbian) que se instalan en el PC de desarrollo. Seleccione las versiones de Qt que desea utilizar para construir el proyecto para cada objetivo.(FIGURA 32)



(FIGURA 32)

Si Qt Creator no puede encontrar una compilación existente para un entorno de desarrollo (versión de Qt) y el objetivo, que comienza desde cero, y crea una nueva versión en el directorio especificado. Qt Creator sugiere un nombre y una ubicación para el directorio que se puede cambiar.

Si se ha construido el proyecto antes, Qt Creator puede utilizar la actual configuración de generación para hacer exactamente la misma construcción que se encuentra en el directorio a disposición de Qt Creator.

Si se sabe que tiene una generación, pero no está en la lista, haga clic en **Agregar Construir** para localizarlo. Seleccione un directorio, y Qt Creator escanea (incluyendo subdirectorios) para las versiones adicionales del proyecto. Qt Creator añade las compilaciones encontradas a la lista de objetivos. También puede editar la configuración de la generación más adelante.

Para abrir un proyecto:

1. Seleccione **Archivo> Abrir o proyecto** y seleccione el proyecto para abrir.
2. En el cuadro de diálogo **Configuración del proyecto,** seleccionar las versiones de Qt para usar como construir objetivos de su proyecto, y haga clic en **Finalizar.**

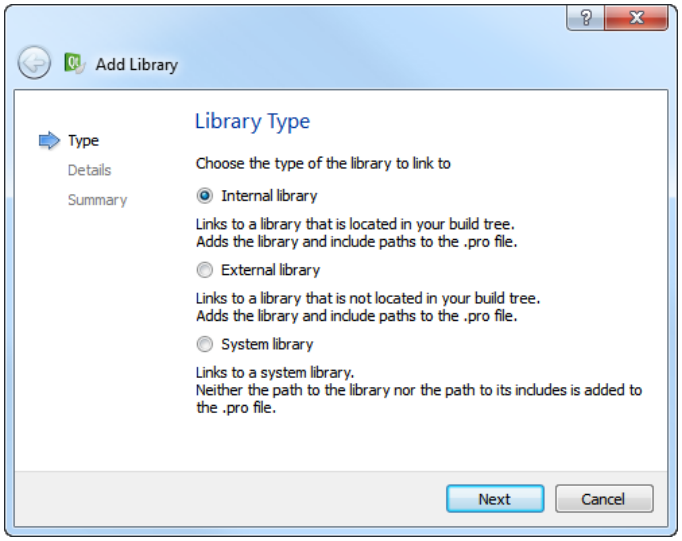
   Nota: Si sólo tiene un entorno de desarrollo instalado, este diálogo se omite.

Qt Creator analiza todos los archivos de origen en el proyecto y realiza un análisis semántico para construir la información que necesita para funciones como la navegación. Una barra de progreso se muestra durante el análisis.

Además de las bibliotecas Qt, puede añadir otras bibliotecas para sus proyectos. La forma en que se agrega la biblioteca depende de si se trata de una biblioteca del sistema o su propia biblioteca o una biblioteca de terceros ubicado en el árbol de construcción del proyecto actual o en otro árbol de construcción.

1. En el panel de **proyectos,** abra el archivo de proyecto (. Pro).
2. Haga clic derecho en el editor de código para abrir el menú contextual y seleccionar **Add Library ....**

Siga las instrucciones del asistente.(FIGURA 33)



(FIGURA 33)

Debido a que las bibliotecas del sistema no suelen cambiar y con frecuencia se encuentran por defecto, no es necesario especificar la ruta a la biblioteca

Por sus propias bibliotecas y bibliotecas terceras, es necesario especificar las rutas. Qt Creator intentará que la ruta de inclusión de una biblioteca sea interna, pero usted tiene que comprobar y modificar si es necesario. Qt Creator añade automáticamente la ruta de inclusión de una biblioteca interna.

Para todas las bibliotecas, seleccione las plataformas de destino de la aplicación, biblioteca o plugin.

Especifique si la biblioteca es estática o dinámica vinculada. Para una biblioteca vinculada estáticamente interna, Qt Creator añade dependencias(PRE_TARGETDEPS) en el archivo del proyecto.

Dependiendo de la plataforma de desarrollo, algunas opciones pueden ser detectados automáticamente. Por ejemplo, en Mac OS, el tipo de biblioteca (Biblioteca o Framework) se detecta de forma automática y la opción está oculta. Sin embargo, si se desarrollan en otra plataforma de Mac OS y quieren construir su proyecto para el sistema operativo Mac, debe especificar el tipo de biblioteca.

El convenio por defecto en Windows es que las versiones de depuración y liberación de una biblioteca tengan el mismo nombre, pero se colocan en diferentes carpetas, por lo general llama *depuración* y de *lanzamiento.* Si la ruta de la biblioteca no contiene cualquiera de estas carpetas, no se puede seleccionar la opción de colocar las bibliotecas en carpetas separadas.

Por otra parte, la letra *d* se pueden añadir al nombre de la biblioteca de la versión de depuración. Por ejemplo, si la versión se llama example.lib, la versión de depuración se llama exampled.lib. Puede especificar que la carta se añade a la versión de depuración y eliminado debido a que la versión de lanzamiento. Si el nombre de la biblioteca termina en *d,* desactive la opción **Quitar "d" sufijo para la** opción **de lanzar la versión.**

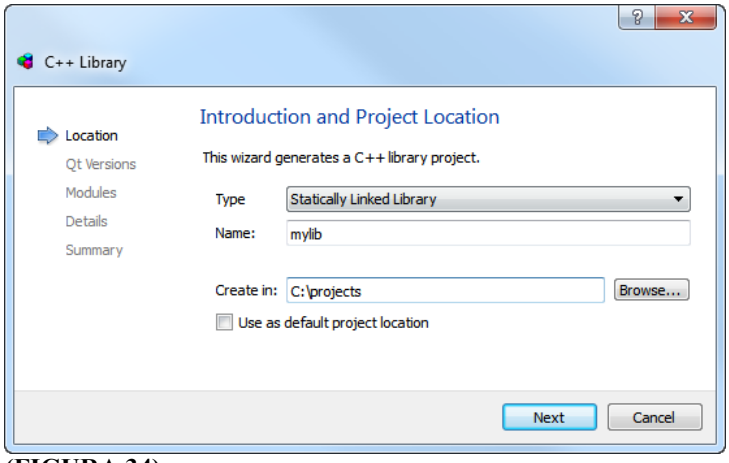
Qt Creator es compatible con la finalización de código y resaltado de sintaxis para las bibliotecas añadido una vez que su proyecto se construye con exito y los enlaces a los mismos.

3.8.1 Ejemplo de Adición de bibliotecas internas:

El siguiente ejemplo muestra cómo agregar una biblioteca vinculada estáticamente al interior de su proyecto.

1. Seleccione **Archivo> Nuevo archivo o proyecto ... > Otros proyectos> C + + Library** para crear la biblioteca.

El cuadro de diálogo **Introduction and project location** es el siguiente:(FIGURA 34)



(FIGURA 34)

2. En el campo **Tipo,** seleccione **Biblioteca enlazados estáticamente.**
3. En el campo **Nombre,** asigne un nombre para la biblioteca. Por ejemplo, **mylib.**
4. Siga las instrucciones del asistente hasta llegar al cuadro de diálogo **Gestión de Proyectos.** En la lista **Agregar a proyecto,** seleccione un proyecto. Por ejemplo, **myapp.**
5. En el panel de **proyectos,** abra el archivo de proyecto (. Pro). Por ejemplo, **myapp.pro.**
6. Haga clic derecho en el editor de código para abrir el menú contextual y seleccionar **Add Library ... > Interno Biblioteca> Siguiente.**
7. En el ámbito de **la Biblioteca,** seleccione **mylib** y haga clic en **Siguiente.**
8. Haga clic en **Finalizar** para agregar la declaración de la biblioteca.

3.9 Gestión de sesiones:

Al salir de Qt Creator, una instantánea de su espacio de trabajo actual se almacena en una *sesión.* Para restaurar la sesión de forma automática al iniciar Qt Creator, seleccione **Archivo> Gestor de sesiones> Restaurar la última sesión en el inicio.**

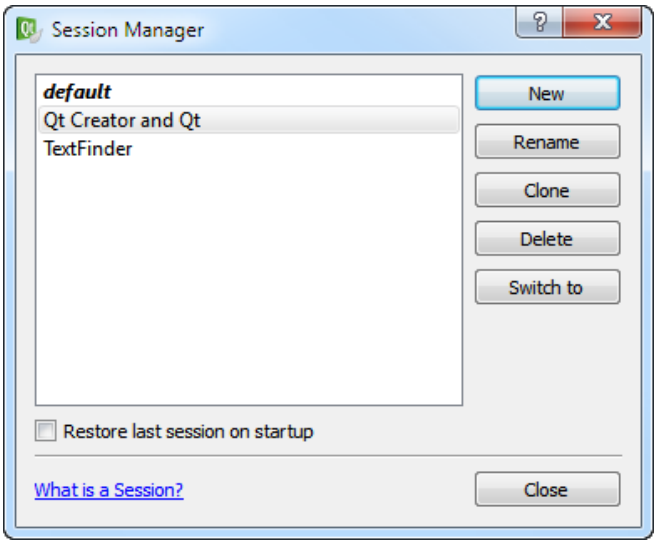
Una sesión es una colección arbitraria de:

- Proyectos de código abierto con sus dependencias (incluidos los proyectos SUBDIRS)
- Editores abiertos
- Puntos de interrupción y expresiones
- Marcadores

Una sesión es personal, es decir, no para ser compartida. No se supone que reflejan la estructura del proyecto. Contiene datos personales, como los marcadores y puntos de interrupción que no suelen ser de interés para los desarrolladores que trabajan en los mismos proyectos.

Por ejemplo, si usted trabaja en un proyecto y la necesidad de cambiar a otro proyecto por un tiempo, usted puede guardar su trabajo como una sesión. Esto hace que sea más fácil volver a trabajar en el primer proyecto posterior.

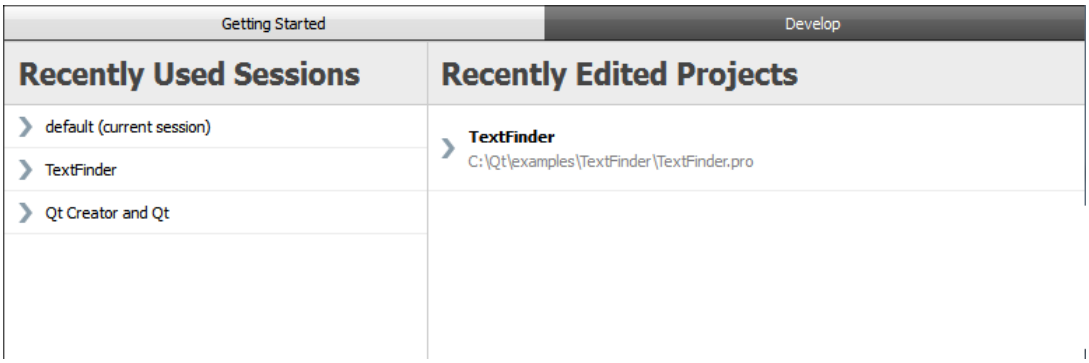
Para crear una nueva sesión o eliminar las sesiones existentes, seleccione **Archivo> Administrador de sesiones**(FIGURA 35)



(FIGURA 35)

Para cambiar entre sesiones, elija **Archivo> Administrador de sesiones.** Si usted no cree o seleccione una sesión, Qt Creator utiliza siempre la sesión por defecto, que se creó la última vez que salió Qt Creator.

Al iniciar Qt Creator, una lista de las sesiones existentes se visualiza en la **pantalla de bienvenida.** (FIGURA 36)



(FIGURA 36)

CAPÍTULO 4

QTCREATOR

Aquí se enseñará a utilizar las herramientas correspondientes al editor de código

Herramientas

- Codificación
- Uso del editor
- Utilizando la barra de herramientas del editor
- La división de la vista del editor
- La división de la vista del editor
- Pasando a Símbolo Definición o declaración
- Usando el modelo de actualización del Código
- Destacando semántica
- Destacando genéricos :
- Destacar y plegables bloques
- Completar Código
- Resumen de los tipos disponibles
- Completar fragmentos de código
- Edición de fragmentos de código
- Agregar y editar fragmentos
- Fragmentos de la eliminación
- Fragmentos de Restablecimiento
- Sangrado del código
- Sangría de archivos de texto
- Sangría de C + + Archivos
- Especificación de la configuración Tab
- Especificar la configuración para el contenido
- Especificar la configuración de las sentencias switch
- Especificación de la alineación

4.0 Codificación

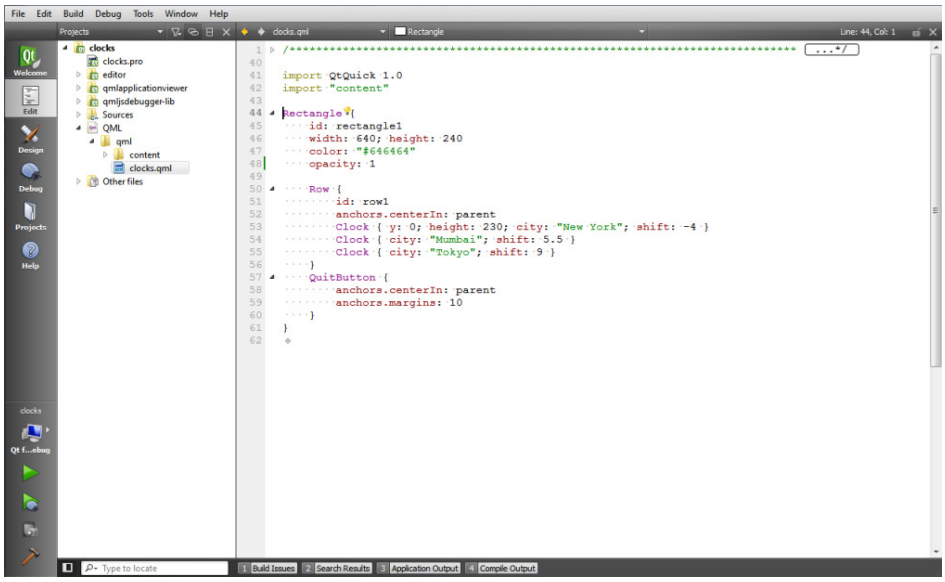
Escribir, editar y navegar en el código fuente es una tarea central en el desarrollo de aplicaciones. Por lo tanto, el editor de código es uno de los componentes clave de Qt Creator. Usted puede utilizar el editor de código en el modo de **edición.**

Las secciones siguientes describen la codificación con Qt Creator:

- **Uso del editor:** describe cómo trabajar en el editor de código, utilice la barra de herramientas del editor, dividió el punto de vista, añadir marcadores y desplazarse por las definiciones de símbolos y declaraciones.
- **Destacando semántica:** describe destacando los elementos de código y bloques, así como el uso de resaltado de sintaxis también para otros tipos de archivos de C + + o QML.
- **Comprobación de la sintaxis del código**: describe cómo los errores se visualizan cuando se escribe código.
- **Completar Código**: describe cómo fragmentos de código y el código se completan los elementos, propiedades, un IDS.
- **Sangrado Código**: describe cómo especificar la sangría de forma global para todos los archivos o por separado para: texto, C + +, o los archivos QML.
- **Buscar y reemplazar**: describe la búsqueda incremental que se destacan las cadenas coincidentes en la ventana mientras se escribe y la de búsqueda avanzada que permite realizar búsquedas de los proyectos actualmente abiertos o archivos en el sistema de archivos. Además, puede buscar símbolos cuando se desea refactorizar código.
- **Refactorización**: se describen las características que le ayudan a mejorar la calidad interna o su aplicación, su rendimiento y extensibilidad, y la legibilidad del código y facilidad de mantenimiento, así como para simplificar la estructura del código.
- **Usando Qt Quick barras de herramientas:** se describe cómo utilizar el Qt Quick Barras de herramientas para editar las propiedades de los elementos QML en el editor de código.
- **Búsqueda con el localizador**: se describe cómo navegar a través de proyectos, archivos, clases, métodos, documentación y sistemas de archivos.
- **Obtención y pegar fragmentos de código:** describe cómo colaborar con otros desarrolladores y el acarreo de pegar fragmentos de código de un servidor.
- **Uso de macros de edición de texto**: se describe cómo grabar y reproducir macros de edición de texto.
- **Configuración del Editor**: se describe cómo cambiar las opciones del editor de texto para adaptarse a sus necesidades específicas.
- **Uso del modo FakeVim:** describe cómo ejecutar el editor principal de una manera similar a la del editor Vim.

4.1 Uso del editor:

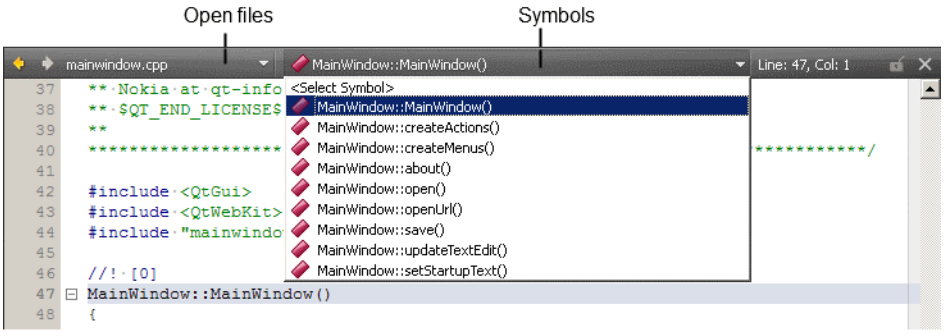
Qt Creator editor de código está diseñado para ayudarle a crear, editar y navegar código. Editor de código de Qt Creator está totalmente equipada con la comprobación de sintaxis, completado de código, ayuda sensible al contexto y en línea, los indicadores de error al teclear.(FIGURA 37)



(FIGURA 37)

4.2 Utilizando la barra de herramientas del editor :

La barra de herramientas del editor se encuentra en la parte superior de la vista del editor. La barra de herramientas del editor se muestra sensible al contexto y los elementos relevantes para el archivo abierto en el editor. (FIGURA 38)



(FIGURA 38)

Utilice la barra de herramientas para navegar entre los archivos abiertos y los símbolos en uso. Para ver hacia adelante o hacia atrás a través de su historial de ubicaciones, haga clic en y .

Para ir a cualquier archivo abierto, seleccionar los **archivos abiertos** en el menú desplegable. Haga clic en el título del menú y seleccione **Copiar ruta completa al portapapeles** para copiar la ruta y el nombre del archivo actual en el portapapeles.

Para saltar a cualquier símbolo que se utiliza en el archivo actual, seleccione una de los **símbolos** en el menú desplegable. Por defecto, los símbolos se muestran en el orden en que aparecen en el archivo. Haga clic en el título del menú y seleccione **Ordenar alfabéticamente** para organizar los símbolos en orden alfabético.

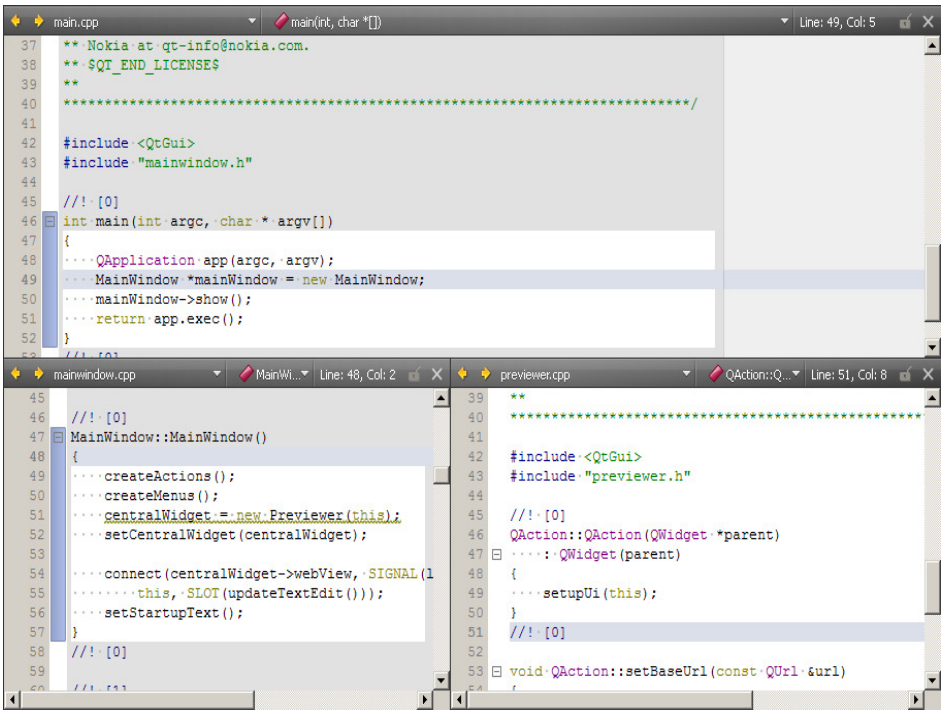
4.3 La división de la vista del editor:

Dividir la vista del editor cuando se quiere trabajar y ver los archivos múltiples en la misma pantalla.

Se puede dividir la vista del editor de la siguiente manera:

- Para dividir la vista del editor en una vista superior e inferior, seleccione **Ventana> Dividir** o presione **Ctrl + E, 2.**

Comando split crea puntos de vista por debajo de la vista del editor activo.(FIGURA 39)



(FIGURA 39)

- Para dividir la vista del editor en vistas adyacentes, seleccione **Ventana>** Side **by Side división** o presione **Ctrl + E, 3.**

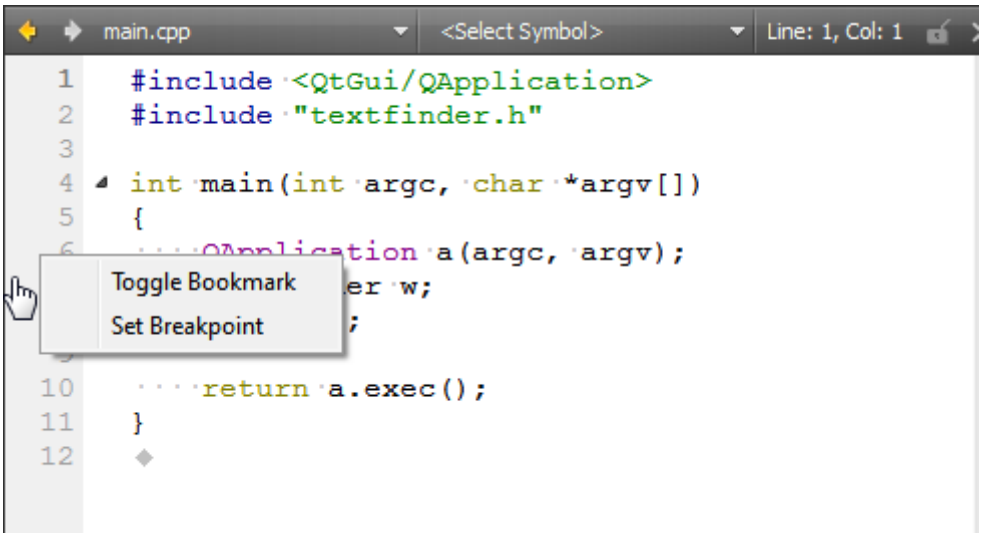
  De lado a lado comando split crea puntos de vista a la derecha de la vista del editor activo.

Para moverse entre las opiniones divergentes, seleccione **Ventana> Ir a la siguiente división** o presione **Ctrl + E, S.**

Para eliminar una vista dividida, coloque el cursor en la vista que desea eliminar y seleccione **Ventana> Quitar división actual** o pulse **Ctrl + S, 0.** Para eliminar todos pero la vista dividida actualmente seleccionado, seleccione **Ventana> Quitar todos los Splits** o presione **Ctrl + E, 1.**

4.4 Utilización de marcadores:

Para insertar o borrar un marcador haga clic en el número de línea y seleccione **Cambiar marcadores** o pulse **Ctrl + M.** (FIGURA 40)



(FIGURA 40)

Para ir al marcador anterior de la sesión actual, presione **Ctrl +,.**

Para ir al marcador siguiente en la sesión actual, presione **Ctrl +..**

Pasando a Símbolo Definición o declaración:

Puede pasar directamente a la definición o la declaración de un símbolo mediante la celebración de las teclas **Ctrl** y haciendo clic en el símbolo.

Para activar esta función en movimiento, en **Herramientas> Opciones ...> Editor de texto> Comportamiento,** seleccione **Habilitar la navegación con el ratón.**

También puede seleccionar el símbolo y pulse **F2,** o haga clic en el símbolo y seleccione **Símbolo Sigue bajo el cursor** para mover a su definición o declaración. Esta característica es compatible con los espacios de nombres, clases, métodos, variables, incluyen estados de cuenta y macros.

Para cambiar entre la definición y declaración de un símbolo, pulse **Mayús + F2** o haga clic en el símbolo y seleccione **Cambiar entre declaración de método / Definición.**

4.5 Usando el modelo de actualización del Código:

Para actualizar la información interna en Qt Creator relacionados con su código, seleccione **Herramientas> C + +> Modelo de código de actualización.**

Nota: En las actualizaciones de indexación Qt Creador del código de forma automática. El uso **de modelos de actualización del Código** sólo como un comando de emergencia.

Destacando semántica:

Qt Creator entiende el C + + y lenguajes QML como código, no como texto sin formato. Se lee el código fuente, lo analiza, y destaca que sobre la base de las comprobaciones semánticas que lo hace para los elementos de código siguientes:

- Tipos (tales como clases, estructuras, y las definiciones de tipo)
- Las variables locales
- Campos de la clase
- Los métodos virtuales

Para especificar el esquema de color a utilizar para poner de relieve semántico, seleccione **Herramientas> Opciones ... > Texto Fuentes> Editor y color.**

Qt Creator soporta resaltado de sintaxis también para otros tipos de archivos de C + + o QML.

4.6 Destacando genéricos :

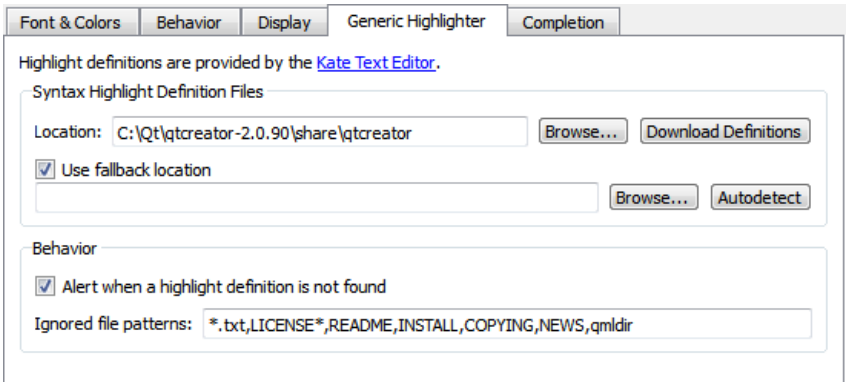
Destacando genérico se basa en archivos de definición de destacar que son proporcionados por el editor de Kate . Puede descargar los archivos de definición de relieve para uso con Qt Creator.

Si usted tiene una instalación de Unix que viene con el editor de Kate, puede que ya han instalado los archivos de definición. Normalmente, los archivos se encuentran en un directorio de sólo lectura, y por lo tanto, usted no puede manejar. Qt Creator se puede tratar de localizar y utilizar como archivos de reserva, cuando la localización primaria no contiene la definición del tipo de archivo actual. También puede especificar el directorio que contiene los archivos de definición preinstalado destacar como la ubicación principal.

Al abrir un archivo para la edición y el editor no puede encontrar la definición para poner de relieve que, aparece una alerta. Usted puede desactivar las alertas. También puede especificar patrones para ignorar los archivos. El editor no le avisará si las definiciones de resalte de los archivos ignorados no se encuentran.

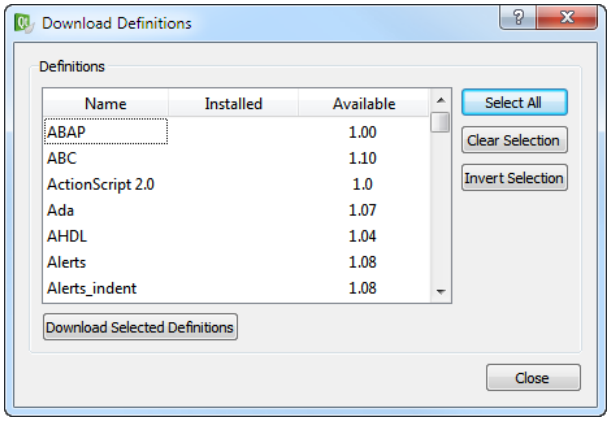
Para descargar los archivos de definición de destacar:

1. Seleccione **Herramientas> Opciones ... > Editor de texto> Resaltador Genérico.** (FIGURA 41)



(FIGURA 41)

2. En el campo **Ubicación,** especifique la ruta de la ubicación principal para los archivos de definición de resaltado.
3. Haga clic en **Descargar Definiciones** para abrir una lista de archivos de definición de destacar disponible para su descarga. (FIGURA 42)

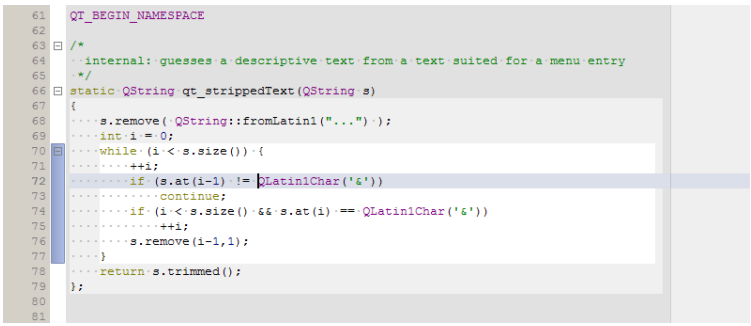


(FIGURA 42)

4. Seleccione los archivos de definición de relieve en la lista y haga clic en **Descargar definiciones seleccionadas.**
5. Seleccione el **lugar de reserva Use** casilla para especificar la ubicación secundaria, donde el editor buscará los archivos de definición de resaltado.
6. Haga clic en **Detectar automáticamente** para permitir que Qt Creator para buscar archivos de definición de destacar en su sistema, haga clic en **Examinar** para localizar en el sistema de archivos usted mismo.
7. En el **archivo** de **patrones de** campo **ignorado,** especificar patrones de ficheros. Usted no recibirá alertas si las definiciones de relieve de los archivos especificados no se encuentran.
8. Haga clic en **Aceptar** para guardar los cambios.

4.7 Destacar y plegables bloques:

Use un bloque para destacar visualmente separar las partes del código que van de la mano. Por ejemplo, cuando se coloca el cursor dentro de las llaves, el código entre llaves se pone de relieve.**(FIGURA 43)**



(FIGURA 43)

Para permitir que destacar bloque, seleccione **Herramientas> Opciones ...> Editor de texto> Pantalla> Seleccione los bloques.**

Use los marcadores de plegado para contraer y expandir bloques de código entre llaves. Haga clic en el marcador plegables para contraer o expandir un bloque. En la figura anterior, los marcadores de plegado se encuentran entre el número de línea y el panel de texto.

Para mostrar los marcadores de plegado, seleccione **Herramientas> Opciones ...> Editor de texto> Pantalla> Mostrar marcadores plegables.** Esta opción está activada por defecto.

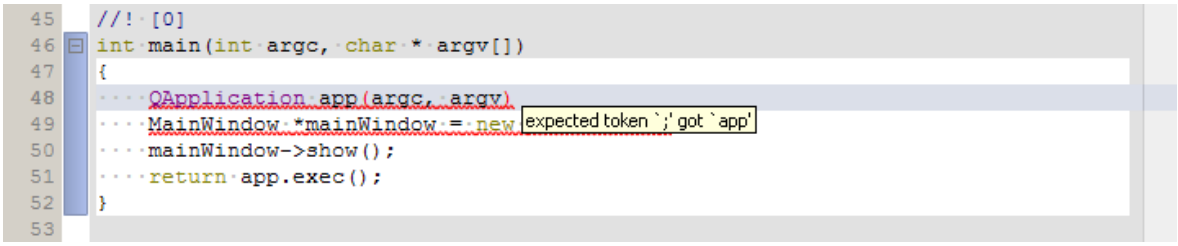
Cuando el cursor se encuentra en un corchete, la llave correspondiente está animada por defecto. Para desactivar la animación, y sólo tienes que seleccionar el bloque y el aparato, seleccione **Herramientas> Opciones ... > Editor de texto> Pantalla** y desactive la opción **Animar paréntesis coincidentes.**

4.8 Comprobación de la sintaxis del código:

Cuando Qt Creator ve a un error de sintaxis en el código que se destaca y muestra los detalles de error cuando usted mueve el puntero del ratón sobre el error.

- Los errores de sintaxis son subrayados en rojo.

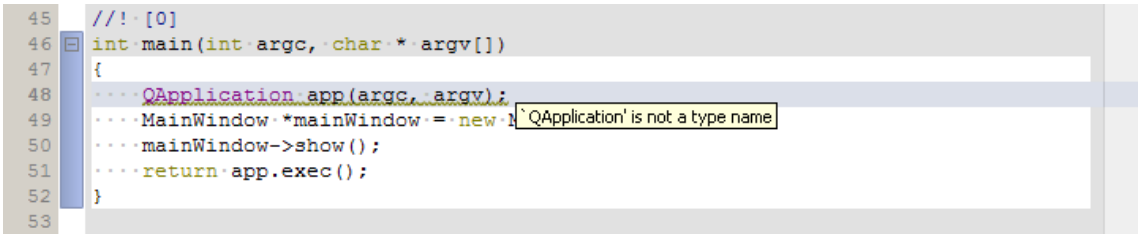
  En la siguiente figura(FIGURA 44), un punto y coma que falta al final de la línea.



(FIGURA 44)

- Errores semánticos y las advertencias se destacó en marrón claro.

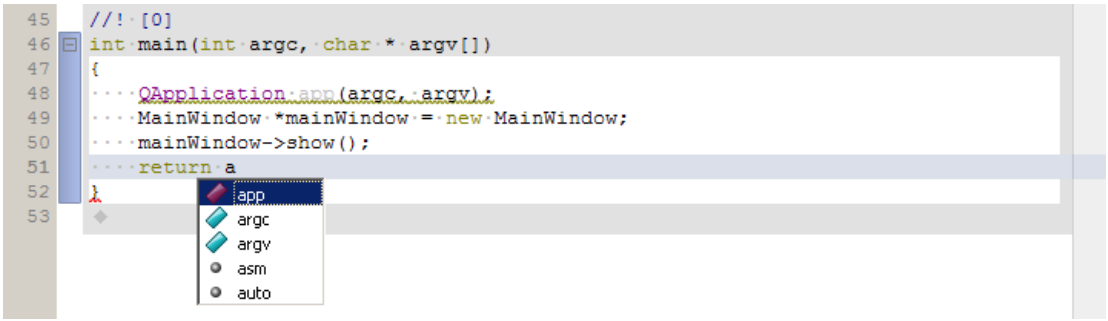
En la figura siguiente(FIGURA 45), el tipo es desconocido.



(FIGURA 45)

4.9 Completar Código :

Como se escribe código, Qt Creator sugiere propiedades, identificaciones, y fragmentos de código para completar el código. Se proporciona una lista de sugerencias sensibles al contexto de la declaración actualmente en el cursor. Pulse la tecla **Tab** o **Enter** para aceptar la sugerencia de seleccionar y completar el código.(FIGURA 46)



(FIGURA 46)

Para abrir la lista de sugerencias en cualquier momento, pulse **Ctrl + Espacio.** Si sólo hay una opción está disponible, Qt Creator se inserta de forma automática.

Al finalizar se invoca de forma manual, Qt Creator completa el prefijo común de la lista de sugerencias. Esto es especialmente útil para las clases con varios miembros del mismo nombre. Para desactivar esta funcionalidad, desactivar **Autocompletar prefijo común** en las preferencias de finalización de código. Seleccione **Herramientas> Opciones ...> Editor de texto> Finalización.**

Por defecto, la finalización de código considera sólo la primera letra mayúsculas y minúsculas. Para aplicar total o no entre mayúsculas y minúsculas, seleccione la opción en el campo de **mayúsculas y minúsculas.**

4.10 Resumen de los tipos disponibles:

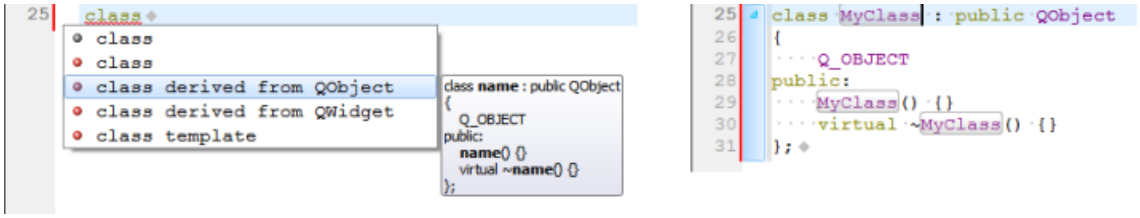
La siguiente tabla muestra los tipos disponibles para completar el código y el icono que se utiliza para cada uno. (Tabla 1)

Icono	Descripción
	Una clase
	Una enumeración
	Un enumerador (valor de una enumeración)
	Una de las funciones
	Una función privada
	Una función de protección
	Una variable
	Una variable privada
	Una variable protegida
	Una señal
	Una ranura
	Un espacio privado
	Un espacio protegido
	Una palabra clave de C + +
	A C + + fragmento de código
	Un elemento QML
	Un fragmento de código QML
	Un macro
	Un espacio de nombres

(Tabla 1)

4.11 Completar fragmentos de código:

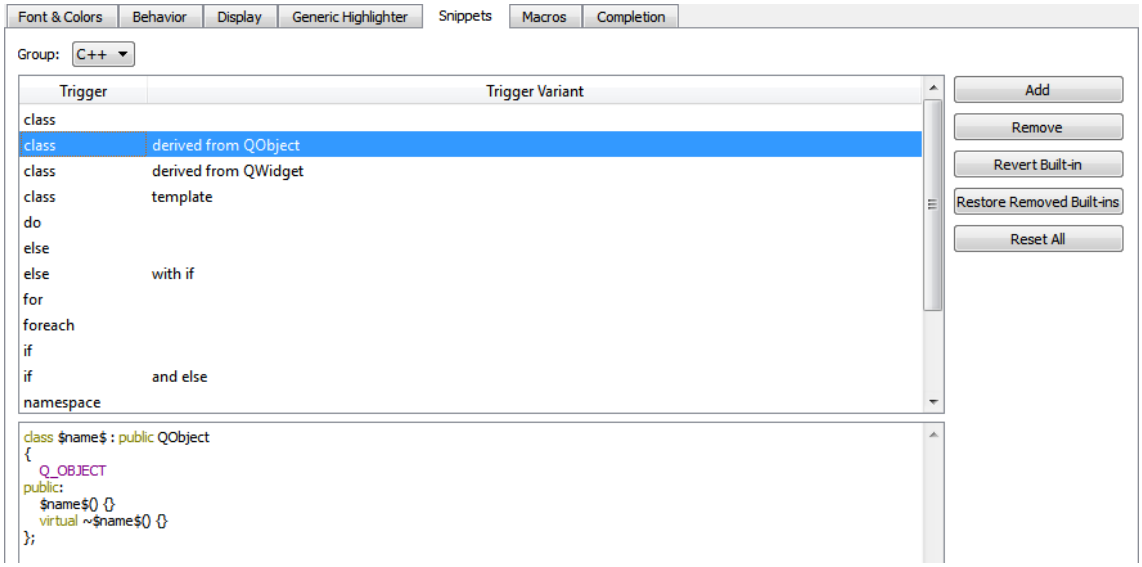
Los fragmentos de código puede consistir en múltiples variables que se especifican los valores . Seleccione un elemento de la lista y pulse **Tab** o **Enter** para completar el código. Presione la tecla **Tab** para moverse entre las variables y especificar los valores para ellos. Cuando se especifica un valor de una variable, todas las instancias de la variable en el fragmento se cambia el nombre. (FIGURA 47)



(FIGURA 47)

4.12 Edición de fragmentos de código:

Los fragmentos de código C + + especifica o construcciones QML código. Puede añadir, modificar y eliminar fragmentos en el editor de fragmentos. Para abrir el editor, seleccione **Herramientas> Opciones ... > Editor de texto> Fragmentos.** (FIGURA 48)



(FIGURA 48)

Qt Creator te ofrece con una función de fragmentos en las siguientes categorías:

- Fragmentos de texto, que puede contener cualquier cadena de texto. Por ejemplo, los comentarios de código
- C + + fragmentos de código, que especificar C + + construcciones de código
- QML de fragmentos de código, que especifican las construcciones de código QML

4.12.1 Agregar y editar fragmentos:

Seleccione un fragmento en la lista para editarlo en el editor de fragmentos. Para agregar un nuevo recorte, seleccione **Agregar.** Especificar un disparador y, si el gatillo ya está en uso, una variante opcional, que aparece en la lista de sugerencias al escribir el código. También se especifica una cadena de texto o C + + o el código QML construir en el editor de fragmentos, en función de la categoría de fragmento. El editor de fragmentos te ofrece:

- Destacando
- Sangría
- Paréntesis coincidentes
- Finalización de código básico

Especificar las variables de los fragmentos en el siguiente formato:

```
Variable $ $
```

Uso exclusivo nombres de las variables dentro de un fragmento, ya que todas las instancias de una variable se cambia cuando se especifica un valor para él.

El editor de fragmentos no comprueba la sintaxis de los fragmentos que modificar o agregar. Sin embargo, al utilizar los fragmentos de código, el editor de código marca los errores con subrayado en rojo.

Para descartar los cambios realizados a un dispositivo integrado en el fragmento, seleccione **Revertir incorporado.**

4.12.2 Fragmentos de la eliminación :

Incorporado en varios fragmentos similares puede referirse a diferentes casos de uso. Para hacer la lista de sugerencias más corto al escribir el código, eliminar los fragmentos integrado que no necesita. Si usted los necesita más tarde, puede restaurarlos.

Para eliminar fragmentos, seleccione un fragmento en la lista, a continuación, seleccione **Quitar.** Para restaurar los fragmentos eliminados, seleccione **Restaurar eliminado empotrados.**

4.12.3 Fragmentos de Restablecimiento :

Para eliminar todos los fragmentos añadidos, y para restaurar todos los fragmentos eliminados, seleccione **Restablecer todo.**

Nota: Si selecciona **en Aceptar** o **Aplicar,** pierde de manera permanente todos sus fragmentos.

4.13 Sangrado del código :

Cuando se escribe código, se sangra de forma automática de acuerdo con el editor de texto seleccionado y las opciones de código de estilo. Seleccione un bloque para guión cuando se pulse la tecla **Tab.** Pulse **Mayús + Tab** para disminuir el sangrado. Puede desactivar la sangría automática.

Cuando se presiona **Retroceso,** la sangría se redujo en un nivel en el espacio en blanco al principio, de manera predeterminada. Usted puede desactivar esta opción.

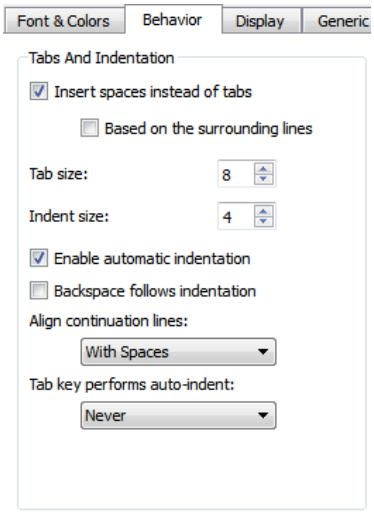
Las líneas de continuación están alineados con la línea anterior mediante el uso de los espacios. Puede desactivar la alineación automática que les sangría a la profundidad lógica. Siempre se puede usar espacios de alineación o espacios de uso o tabuladores en función de las otras opciones que haya seleccionado.

Puede especificar la sangría de forma global para todos los archivos o por separado para:

- Los archivos de texto
- C + + archivos
- QML archivos

Puede especificar la sangría de forma global para todos los archivos de un tipo particular o por separado para cada proyecto.

4.14 Sangría de archivos de texto :



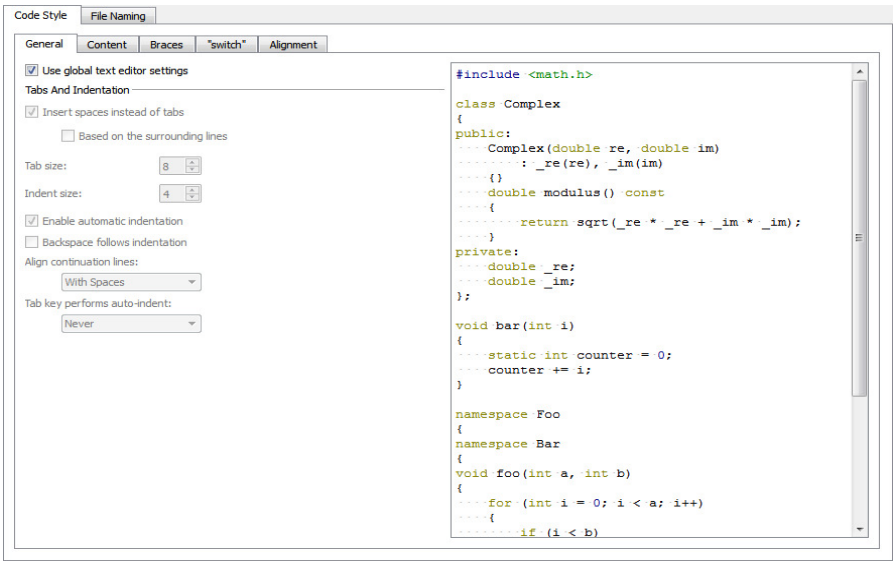
Para especificar la configuración global de sangría para el editor de texto, seleccione **Herramientas> Opciones ... > Editor de texto> Comportamiento.** También puede utilizar estos valores a nivel mundial para todos los editores y los archivos. (FIGURA 49)

Para especificar la configuración de un proyecto en particular, seleccione **Proyectos> Editor de Configuración.**

(FIGURA 49)

4.15 Sangría de C + + Archivos :

Para especificar la configuración global de sangría para el C + + editor, seleccione **Herramientas> Opciones ... > C + +.** (FIGURA 50)



(FIGURA 50)

Para especificar la configuración de un proyecto en particular, seleccione **Proyectos> Código de ajustes de estilo.**

Puede especificar cómo:

- Interpretar la **ficha** y presiona tecla de **retroceso.**
- Guión del contenido de las clases, métodos, bloques y espacios de nombres.
- Guión llaves en las clases, espacios de nombres, las enumeraciones, métodos, y los bloques.
- Control de las sentencias switch y su contenido.
- Alinear las líneas de continuación.

Puede utilizar la vista previa en vivo para ver las opciones de cambiar la sangría.

4.16 Especificación de la configuración Tab:

Usted puede especificar la configuración de la ficha en los siguientes niveles:

- Configuración global para todos los archivos
- Mundial de C + + valores para C + + archivos
- La configuración global de Qt Quick para los archivos QML
- Configuración del proyecto específico para todos los editores de los archivos en el proyecto
- Configuración del proyecto específico para C + + archivos del proyecto
- Configuración del proyecto específico para archivos QML en el proyecto

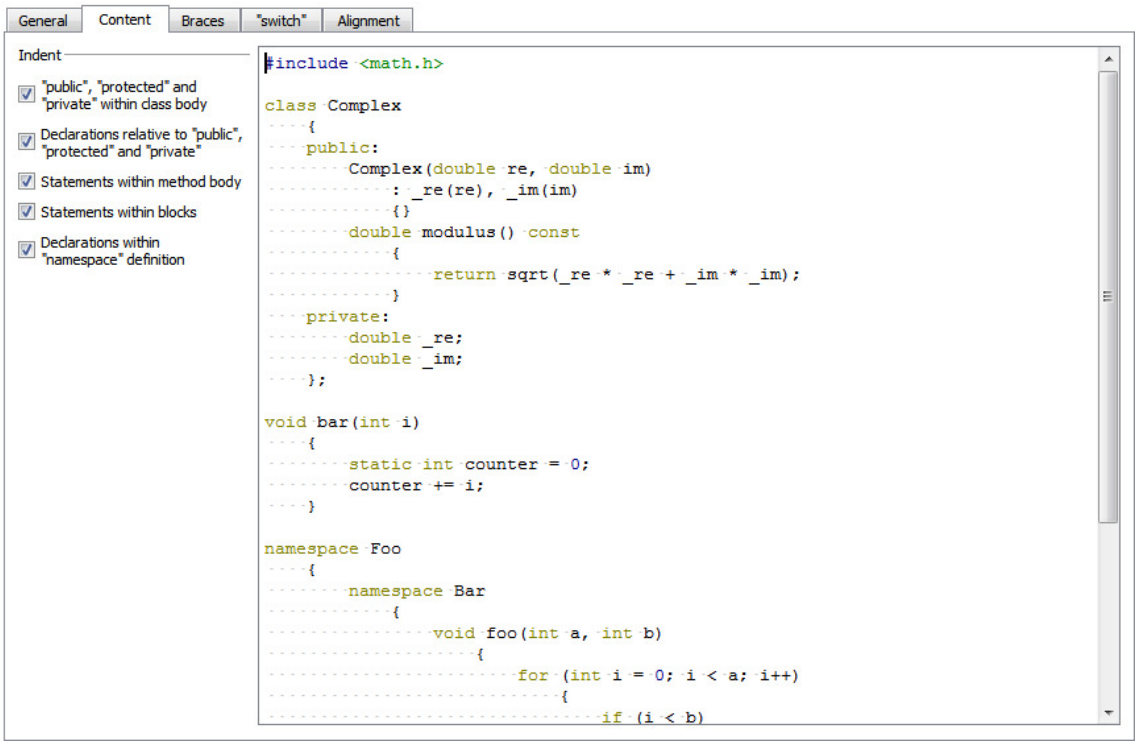
De manera predeterminada, la ficha de longitud en el editor de código es de 8 espacios. Puede especificar la longitud de la ficha por separado para cada proyecto y para diferentes tipos de archivos.

El editor de código también puede determinar si tabuladores o espacios se utilizan en la línea siguiente o anterior y copie el estilo.

La tecla **Tab** automáticamente la sangría del texto cuando se presiona, o sólo cuando el cursor se encuentra dentro de los espacios en blanco.

4.17 Especificar la configuración para el contenido:

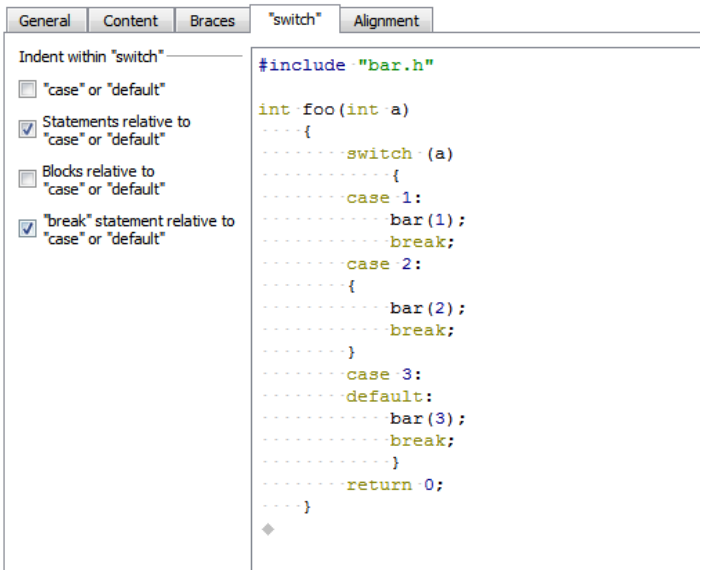
Puede hacer declaraciones públicas, protegidos y privados y las declaraciones relacionadas con ellos dentro de las clases. (FIGURA 51)



(FIGURA 51)

4.18 Especificar la configuración de las sentencias switch :

Puede guión declaraciones de casos o por defecto, o declaraciones o bloques relacionados con ellos en las instrucciones switch. (FIGURA 52)

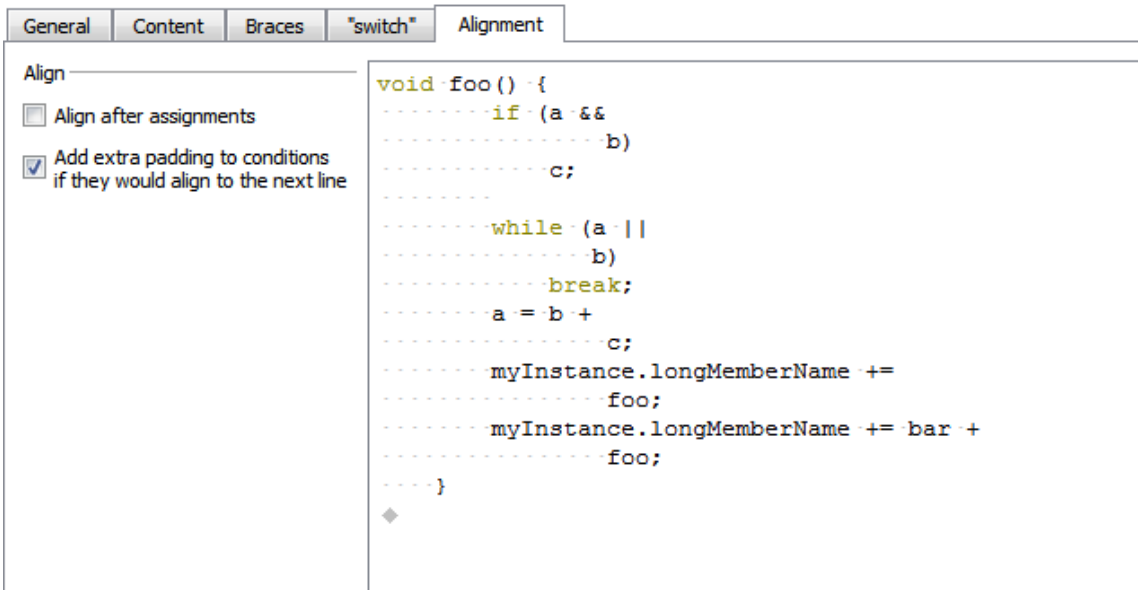


(FIGURA 52)

4.19 Especificación de la alineación :

Para alinear las líneas de continuación de fichas después de las tareas, tales como = o = +, seleccione **Alinear las tareas después de** la casilla de verificación. Puede especificar opciones adicionales para la alineación de las líneas de continuación en la ficha **General.**

También puede agregar espacios para declaraciones condicionales, por lo que no están alineados con la línea siguiente. Por lo general, esto sólo afecta a `las` sentencias *if*. (FIGURA 53)



(FIGURA 53)

CAPÍTULO 5

Integrando opengl en Qt creador

En este capítulo se darán los conceptos básicos para integrar OpenGl en QtCreator, y los pasos aseguir para realizar dicho propósito.

5.0 Integrando opengl en Qt creator

El módulo de Qt OpenGL hace que sea fácil de usar OpenGL en aplicaciones Qt. Proporciona una clase widget OpenGL que puede ser utilizado como cualquier otro widget de Qt, excepto que se abre un búfer de pantalla OpenGL en la que se puede utilizar la API de OpenGL para representar el contenido.

El interfaz de una aplicación OpenGL debe ser creado con otro conjunto de herramientas, tales como X platform, Microsoft Foundation Classes (MFC) under Windows, or Qt en ambas plataformas.

Para incluir las definiciones de las clases del módulo, se debe usar la siguiente directiva(Tabla 2):

```
#include <QtOpenGL>
```

(Tabla 2)

Para enlazar con el módulo, agregue esta línea al archivo qmake.pro(Tabla 3):

```
QT += opengl
```

(Tabla 3)

El módulo de Qt OpenGL se implementa como una plataforma independiente, Qt / C + + con envoltura alrededor de la GLX dependiente de la plataforma (versión 1.3 o posterior), WGL, o AGL C API. Aunque la funcionalidad básica siempre es muy similar a la marca de la biblioteca GLUT, las aplicaciones que utilizan el módulo de Qt OpenGL puede tomar ventaja de toda la API de Qt para la no funcionalidad de OpenGL específica de la interfaz gráfica de usuario.

El módulo de QtOpenGL es parte de la edición Qt marco completo y las versiones de código abierto de Qt. Está disponible en Windows, X11 y Mac OS X. Qt for Embedded Linux es compatible con OpenGL ES (OpenGL for Embedded Systems). Para poder utilizar la API de OpenGL en Qt para Embedded Linux, que debe estar integrado con el sistema de ventanas Q (QWS).

5.0.1 Instalación

Al instalar Qt para X11, el script de configuración se detectará si los encabezados de OpenGL y las bibliotecas se instalan en su sistema, y si es así, que incluirá el módulo QtOpenGL en la biblioteca Qt. (Si las cabeceras OpenGL o las bibliotecas se encuentran en un directorio no estándar, puede que tenga que cambiar el QMAKE_INCDIR_OPENGL y / o QMAKE_LIBDIR_OPENGL en el archivo de configuración de su sistema).

Al instalar Qt para Windows y Mac OS X, el módulo QtOpenGL siempre se incluye.

5.0.2 Cómo utilizar superposiciones X11 con Qt

Superposiciones de X11 son un mecanismo poderoso para las anotaciones de dibujo, etc, en la parte superior de una imagen sin destruirla, lo que ahorra una gran cantidad de imágenes en tiempo de renderizado.

La extensión de OpenGL Qt incluye soporte directo para el uso de superposiciones OpenGL. Para muchos usos de plantillas, esto hace que la técnica descrita a continuación sea redundante. La siguiente es una discusión sobre el uso QGL widgets en planos superpuestos.

En el caso típico, las superposiciones de X11 puede ser utilizado junto con la versión actual de Qt y la extensión de OpenGL Qt. Los requisitos son los siguientes:

Su servidor X y tarjeta gráfica / hardware debe ser compatible con superposiciones. Para muchos servidores X, soporte de superposición se puede activar con una opción de configuración.
El servidor X debe (ser configurado para) usar la superposición visual por defecto visual. Servidores X más modernos pueden hacerlo, ya que este tiene la ventaja añadida de que los menús emergentes, la superposición de ventanas, etc, no afectará a las imágenes subyacentes en el plano principal, evitando de este modo se vuelve a dibujar las caras.
El mejor (más profundo) visual para la representación de OpenGL es en el plano principal. Normalmente, los servidores X que se superpone proporcionan un apoyo de 24-bit de color verdadero visual en el plano principal, y un PseudoColor de 8-bits (por defecto) visual en el plano de superposición.

Se puede utilizar todas las capacidades de dibujo de QPainter para dibujar anotaciones, etc.

Un plano de superposición tiene un color específico llamado el color transparente. Los Píxeles dibujados en este color no serán visible, sino que la imagen subyacente se mostrará a través de OpenGL.

Para utilizar esta técnica, no debe utilizar el *QApplication:: ManyColor o QApplication:: especification* de color verdadero ya que esto obligará a los widgets de Qt normales utilizar un visual TrueColor, que normalmente será en el plano principal, no en el plano de superposición que desee.

5.0.3 QGL Referencia del Espacio de nombres:

El espacio de nombres QGL especifica los diversos identificadores utilizados en el módulo de Qt OpenGL.(Tabla 4)

```
#include <QGL>
```

(Tabla 4)

TIPOS:

- *enum* ***FormatOption*** *{ DoubleBuffer, DepthBuffer, Rgba, AlphaChannel, ..., NoSampleBuffers }*
- *flags* ***FormatOptions***

El módulo QtOpenGL ofrece clases que hacen que sea fácil de usar OpenGL en aplicaciones Qt.(Tabla 5),(Tabla 6).

5.0.4 Los espacios de nombres:

QGL	Especifica los diversos identificadores utilizados en el módulo de Qt OpenGL

(Tabla 5)

Classes

QGLColormap	Se utiliza para la instalación de mapas de colores personalizados en QGLWidgets
QGLContext	Encapsula un contexto OpenGL
QGLFormat	Especifica el formato de visualización de un contexto OpenGL
QGLFramebufferObject	Encapsula un objeto framebuffer OpenGL
QGLPixelBuffer	Encapsula un pBuffer OpenGL
QGLWidget	Widget para la representación de gráficos OpenGL

(Tabla 6)

5.0.5 Descripción detallada

El espacio de nombres QGL especifica los diversos identificadores utilizados en el módulo de Qt OpenGL.

Tipo de documentación:

enum QGL::FormatOption
flags QGL::FormatOptions

Esta enumeración especifica las opciones de formato que se puede utilizar para configurar un contexto OpenGL. Estas se establecen con

QGLFORMAT::SETOPTION();

Constant	Value	Description
QGL::DoubleBuffer	0x0001	Especifica el uso de doble buffer.
QGL::DepthBuffer	0x0002	Activa el depth buffer.
QGL::Rgba	0x0004	Especifica que el contexto debe usar RGBA como su formato de píxel.
QGL::AlphaChannel	0x0008	Activa el uso de un canal alfa
QGL::AccumBuffer	0x0010	Activa el buffer de acumulación
QGL::StencilBuffer	0x0020	Permite el uso de una plantilla
QGL::StereoBuffers	0x0040	Permite el uso de un búfer estéreo para su uso con hardware de visualización.
QGL::DirectRendering	0x0080	Especifica que el contexto que se utiliza para la prestación directa a una pantalla.
QGL::HasOverlay	0x0100	Permite el uso de una plantilla
QGL::SampleBuffers	0x0200	Permite el uso de tampones de muestra.
QGL::SingleBuffer	DoubleBuffer<<16	Especifica el uso de un único buffer, en lugar de tampones doble.
QGL::NoDepthBuffer	DepthBuffer<<16	Deshabilita el uso de un búfer de profundidad..
QGL::ColorIndex	Rgba<<16	Especifica que el contexto debe utilizar un índice de color como su formato de píxel..
QGL::NoAlphaChannel	AlphaChannel<<16	Deshabilita el uso de un canal alfa.
QGL::NoAccumBuffer	AccumBuffer<<16	Deshabilita el uso de un buffer.
QGL::NoStereoBuffers	StereoBuffers<<16	Deshabilita el uso de tampones estéreo.
QGL::IndirectRendering	DirectRendering<<16	Especifica que el contexto se utiliza para la prestación indirecta a un búfer.
QGL::NoOverlay	HasOverlay<<16	Deshabilita el uso de una plantilla.
QGL::NoSampleBuffers	SampleBuffers<<16	Deshabilita el uso de tampones de muestra.

(Tabla 7)

5.0.6 La clase QGLWidget.

QGLWidget proporciona funcionalidad para la visualización de gráficos OpenGL integrado en una aplicación Qt. Es muy fácil de usar. Se hereda de él y utiliza la subclase como cualquier QWidget, excepto que tiene la opción entre el uso de QPainter y comandos estándar OpenGL.

QGLWidget proporciona tres funciones prácticas virtuales que se pueden implementar de nuevo en la subclase para realizar las tareas típicas de OpenGL:

paintGL ()

- Representa la escena OpenGL. Se llama cada vez que el widget necesita ser actualizado.

resizeGL ()

- Establece la ventana gráfica OpenGL, proyección, etc se llama cada vez que el flash ha cambiado el tamaño (y también cuando se muestra por primera vez, ya que todos los reproductores de nueva creación tiene el evento cambiar automáticamente el tamaño).

initializeGL ()

- Establece el contexto de renderizado OpenGL, define las listas de la pantalla, etc se llama una vez antes de la resizeGL primera vez () o paintGL () es llamado.

Aquí está una idea aproximada de cómo puede ser una subclase de QGLWidget, (FIGURA 54),:

```
class MyGLDrawer : public QGLWidget
 {
     Q_OBJECT        // must include this if you use Qt signals/slots

 public:
     MyGLDrawer(QWidget *parent)
         : QGLWidget(parent) {}

 protected:

     void initializeGL()
     {
         // Set up the rendering context, define display lists etc.:
         ...
         glClearColor(0.0, 0.0, 0.0, 0.0);
         glEnable(GL_DEPTH_TEST);
         ...
     }

     void resizeGL(int w, int h)
     {
         // setup viewport, projection etc.:
         glViewport(0, 0, (GLint)w, (GLint)h);
         ...
         glFrustum(...);
         ...
```

```
    }

    void paintGL()
    {
        // draw the scene:
        ...
        glRotatef(...);
        glMaterialfv(...);
        glBegin(GL_QUADS);
        glVertex3f(...);
        glVertex3f(...);
        ...
        glEnd();
        ...
    }

};
```

(FIGURA 54)

Si se necesita iniciar un repintado de otros lugares que paintGL () (un ejemplo típico es cuando se utiliza temporizadores para animar las escenas), se debe llamar a la función updateGL del widget () .

El contexto del widget de renderizado de OpenGL se actualiza cuando paintGL (), resizeGL (), o initializeGL () es llamado. Si es necesario llamar el las funciones de la API del estandar de OpenGL de otros lugares (por ejemplo, en el constructor del widget o en las funciones propias pinturas), se debe llamar a makeCurrent () en primer lugar.

QGLWidget proporciona funciones para solicitar un nuevo formato de pantalla y también se pueden crear widgets con los contextos de representación personalizada.

También se puede compartir listas de visualización OpenGL entre QGLWidgets

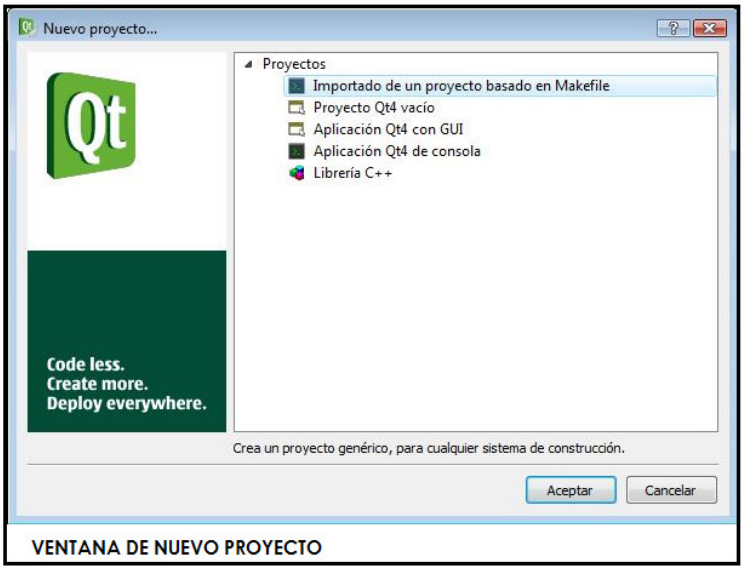
Hay que tener en cuenta que en Windows, la pertenencia a un QGLContext QGLWidget tiene que ser recreado en la QGLWidget del padre. Esto es necesario debido a las limitaciones en la plataforma Windows. Lo más probable es causar problemas para los usuarios que han instalado sus subclases y QGLContext propia en un QGLWidget. Es posible solucionar este problema poniendo el QGLWidget dentro de un widget simulado y luego una reparentalización del widget ficticia, en lugar de la QGLWidget. Lo mejor es esquivar el tema por completo,a no ser que necesite esta funcionalidad.

5.1 PASOS PARA INTEGRAR OPENGL EN QTCREATOR

1. Cuando se abre el programa QtCreator la primera ventana que a aparece es la de la figura (FIGURA 55). Aquí se ha de escoger la opción que aparece en la inferior derecha : “crear nuevo proyecto”.

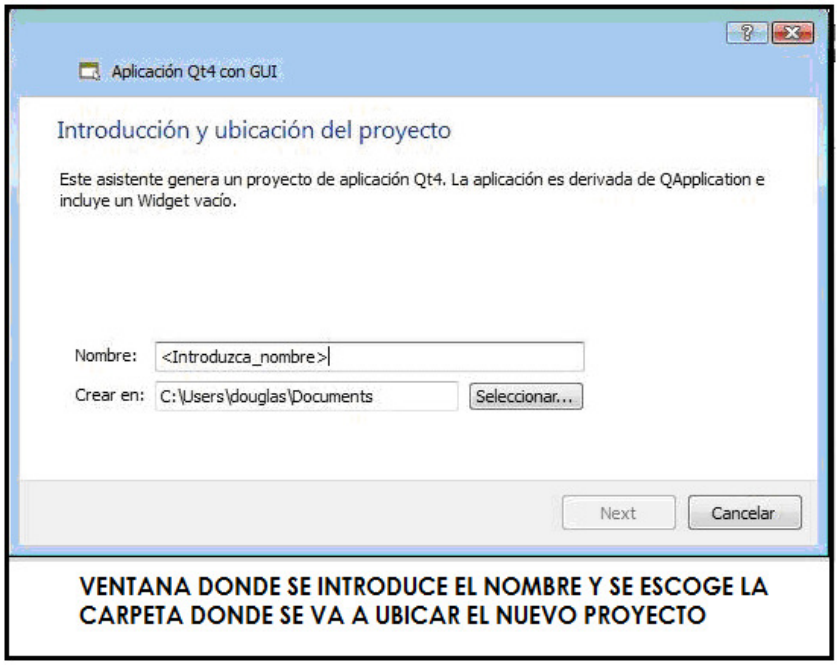
(FIGURA 55)

2. La ventana siguiente figura (FIGURA 56) enseña los distintos proyectos que se pueden crear en QtCreator.
   Aquí se ha de escoger la opcioón Aplicación Qt4 con GUI para poder desarrollar la interfaz gráfica de usuario. Consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo del ordenador.



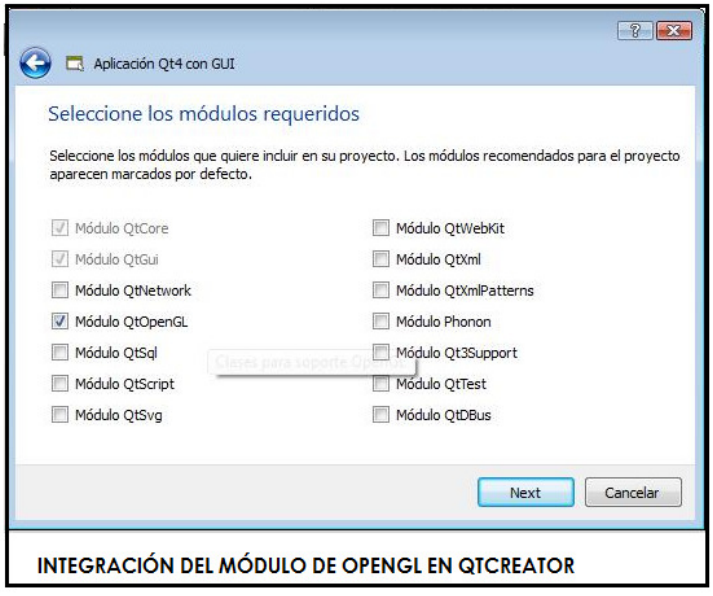
(FIGURA 56)

3. En la ventana siguiente figura (FIGURA 57)que aparece se ha de ingresar el nombre del proyecto nuevo y la ubicación de este



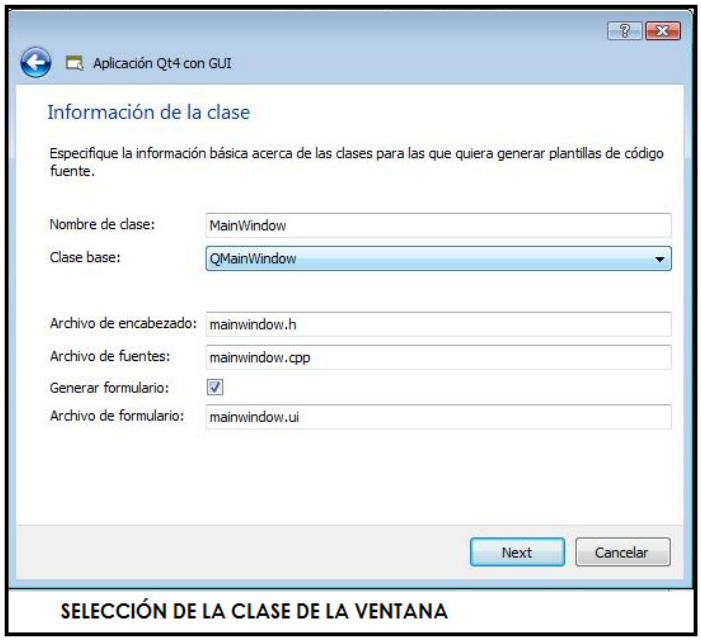
(FIGURA 57)

4. En esta ventana (FIGURA 58)la parte mas importante de la integración de OpenGl en QtCreator ya que aquí es donde se agrega dicho módulo.



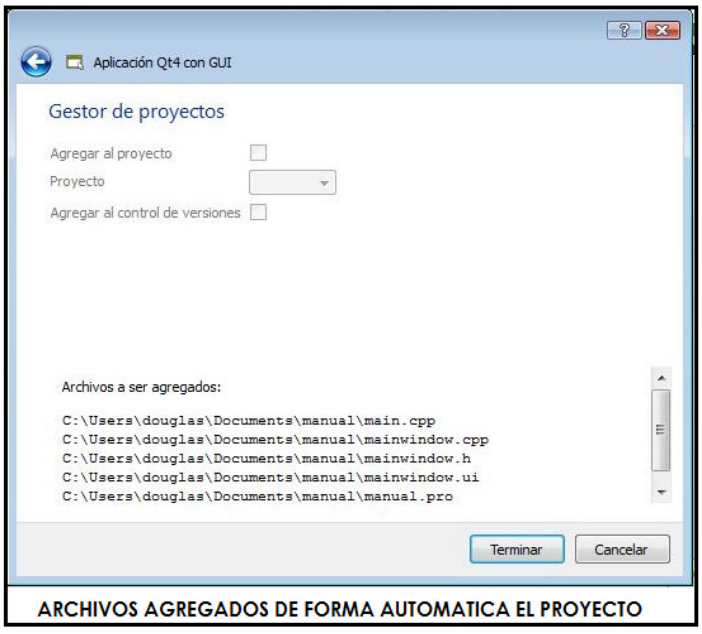
(FIGURA 58)

5. En la ventana siguiente (FIGURA 59) se ha de escoger la clase a la que quiere que corresponda Widget.



(FIGURA 59)

6. La ventana siguiente (FIGURA 60) se muestra los archivos que se agregan de forma predeterminada y automática al proyecto

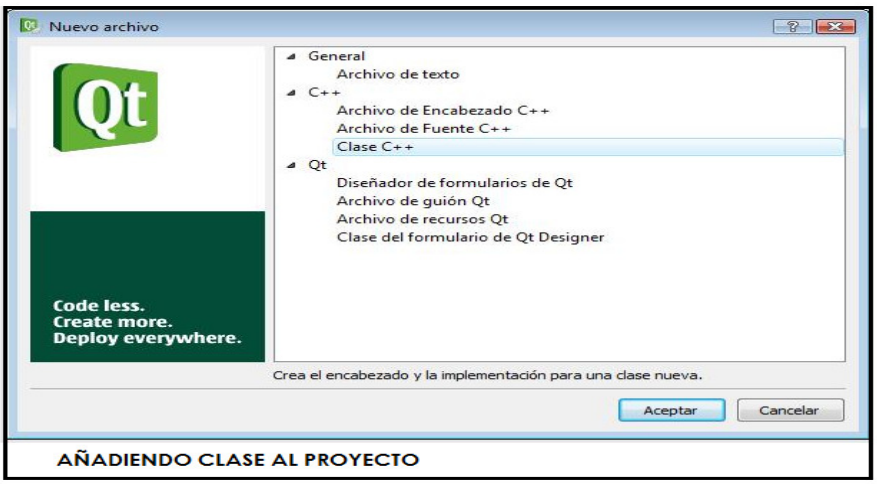


(FIGURA 60)

7. La siguiente ventana (FIGURA 61) que corresponde al navegador de archivos del proyecto , en la misma se aprecia los archivos generados de forma automática a nuestro proyecto.
   Para nuestro proyecto hemos de eliminar el archivo mainwindow.ui.

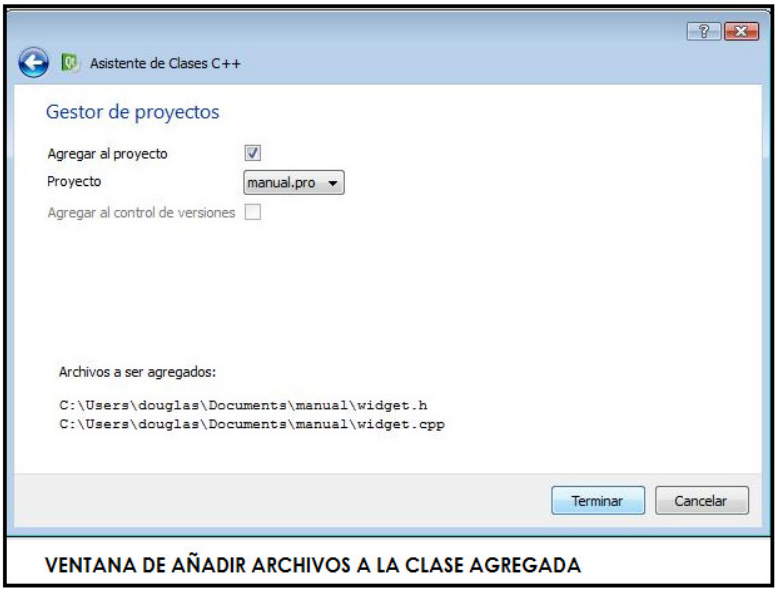
(FIGURA 61)

8. Al proyecto se ha de añadir archivos mas del tipo Clase C++ mediante la opción agregar archivos al proyecto que se ha visto en el capítulo anterior. Para cual aparece la siguiente ventana(FIGURA 62)



(FIGURA 62)

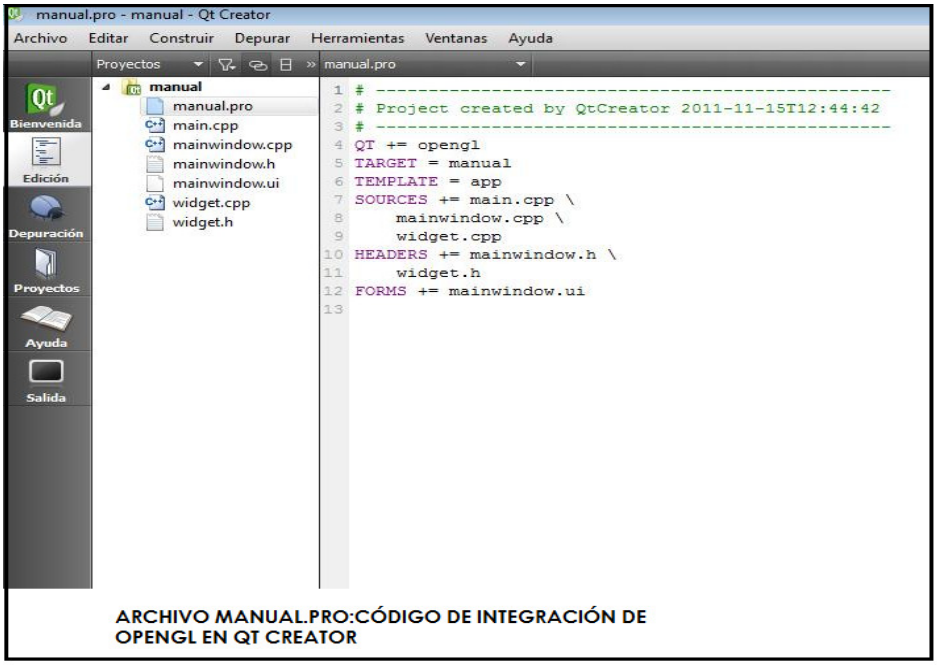
9. La siguiente ventana (FIGURA 63) muestra los archivos que se añaden de forma automática y predeterminada a la clase añadida



(FIGURA 63)

5.1.1 Manual.pro

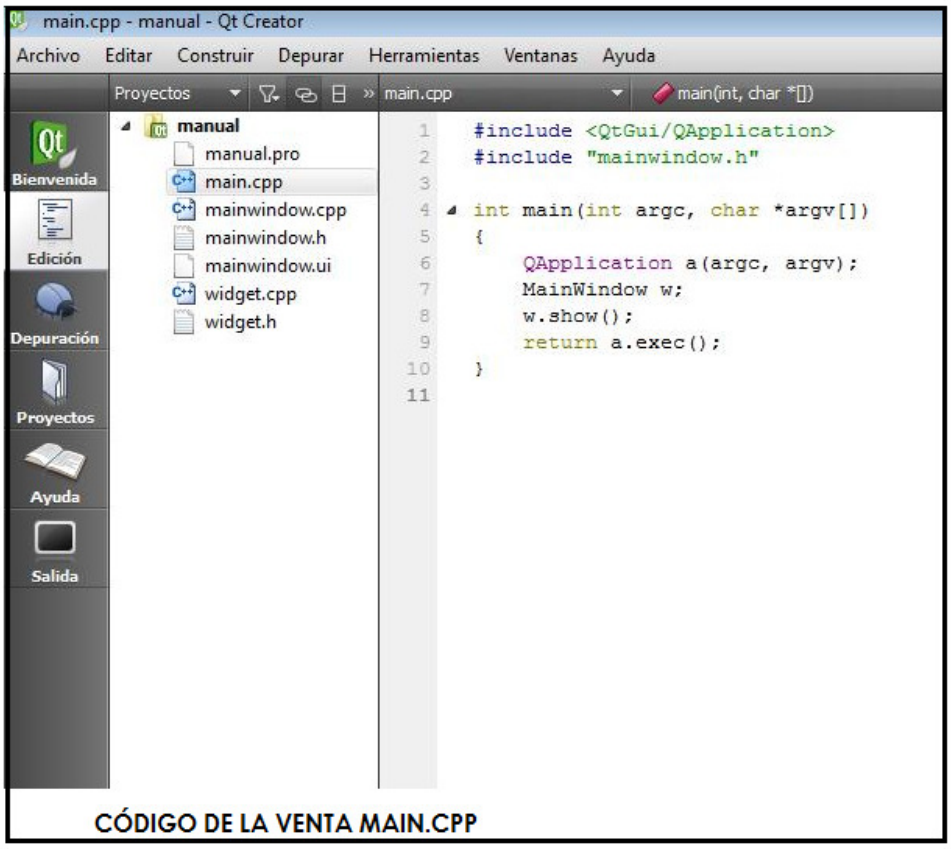
La siguiente ventana (FIGURA 64) muestra los archivos creados y agregados al proyecto anteriormente. En cada uno de estos archivos se crean los códigos necesarios para hacer uso de OpenGl en QtCreator. Como es el caso del archivo "manual.pro".



(FIGURA 64)

5.1.2 Main.cpp

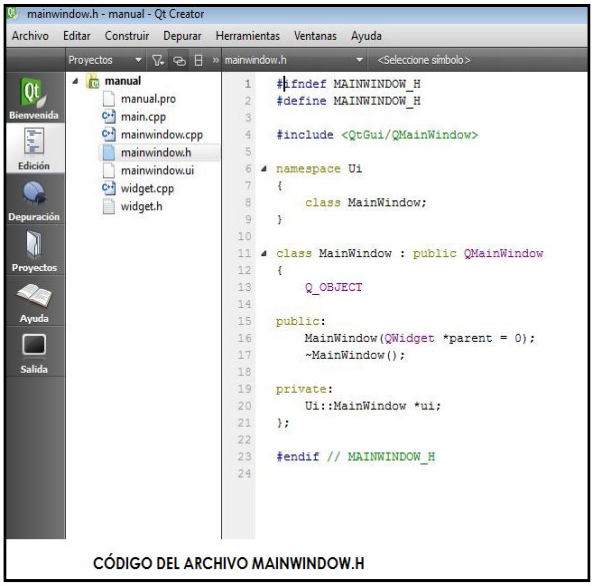
En esta ventana (FIGURA 65) se muestra el código correspondiente al archivo main.cpp, donde se atribuye el manejador de la ventana windows



(FIGURA 65)

Se puede apreciar en el editor de código que se incluye la librería <QtGui/QApplicattion>, y la librería correspondiente a "mainwindow.h"

5.1.3 Mainwindow.h



En este archivo (FIGURA 66) es donde se inserta el código necesario para desarrollar la clase del GLWidget y los atributos de este objeto.

(FIGURA 66)

5.1.4 Wiget.h

En esta ventana (FIGURA 67) se puede apreciar el código base correspondiente para crear la clase de un objeto y es aquí donde se conectan las señales con los slots, además, es aquí donde se debe hacer la llamada a las funciones tales como :

Void initiaizaGL();
Void SaintGL();
Void ResizeGL(int with,int height);
*Void mausepressevent(Q.Mouseevent*event);*
*Void mouseMoveEvent(QMoveevent*event);*

Estas funciones son las que permiten desarrollar el diseño en tres dimensiones con OpenGL.

(FIGURA 67)

Es en este archivo donde se debe insertar la función *GLuint make Object()* y *GLuint object* que se encarga de crear la ventana GlWidget

CAPÍTULO 6

INTRODUCCIÓN: USO DE OPENGL

OpenGL no es un lenguaje de programación ; es un API (*Application Programmig Interfaces*). Un programa basado en OpenGL está escrito en algún lenguaje de programación (como C o C++) y hace llamadas a una o más librerías de OpenGL.

Como API, la librería OpenGL sigue la convención de llamadas de C. En este tutorial, los ejemplos estan escritos en C y estan diseñados para ejecutarse bajo Windows .

OpenGL es una API sofisticada y muy eficaz para crear gráficos 3D, con mas de 300 comandos que cubren todo. OpenGl no tiene una sola función o comando relacionada con la administración de ventanas o de la pantalla. Asimismo, no tiene funciones para entrada de teclado o interacciones con el ratón. Se puede crear una ventana distinta en plataformas diferentes. Es aquí donde interviene el programa Qt Creator.

6.0 TIPOS DE DATOS EN OPENGL

Para que se facilite la portabilidad del código OpenGL de una plataforma a otra, OpenGL define sus propios tipos de datos. Estos tipos de datos se trazan a tipo de datos C normalmente que podemos usar en su lugar, si así se desea. Sin ambargo los diversos compiladores y entornos tienen sus propias reglas para el tamaño y el diseño de memoria para las diversas variables C. Al usar los tipos devariables definidos por OpenGL, podemos aislar el código de estos tipos de cambio.

Los tipos de datos se corresponden con tipos de datos normales en C que podemos usar en su lugar si queremos. La siguiente tabla recoge los tipos de datos de OpenGL, su correspondencia con los tipos de datos de C y el sufijo apropiado para los literales (Tabla 8):

Tipo de datos OpenGL	Representación interna	Definición como tipo C	Sufijo literal en C
GLbyte	Entero de 8 bits	Char con signo	d
GLshort	Entero de 16 bits	Short	s
GLint, GLsizei	Entero de 32 bits	long	l
GLfloat, GLclampf	Coma flotante de 32 bits	Float	f
GLdouble, GLclampd	Coma flotante de 64 bits	double	d
GLubyte, GLboolean	Entero de 8 bits sin signo	Unsigned char	ub
GLushort	Entero de 16 bits sin signo	Unsigned short	us
GLuint, GLenum, GLbitfield	Entero de 32 bits sin signo	Unsigned long	us

(Tabla 8)

Todos los tipos de datos comienzan con GL para denotar OpenGL a la mayoría les sigue su correspondiente tipos de datos C(byte, short, int, float, ,etc.).

Algunos tienen primero una u para denotar un tipo de datos sin firmar, como ubyte que denota un byte sin firmar.

Los punteros y matrices no tienen una consideración especial. Una matriz de diez variables GLshort se declararía como:

GLshort matriz[10]

Y una matriz de diez punteros a variables GLdouble se declararía como:

*GLdouble *matriz[10]*

6.1 CONVENIO DE DENOMINACION DE FUNCIONES

Las funciones siguen una convención para denominarse que nos indica de qué biblioteca es la función y a menudo cuántos argumentos y de qué tipo usa la función.

Todas las funciones tienen una raíz que representa el comando OpenGL que le corresponde. Por ejemplo, la función **glColor3f()** tiene la raíz Color. El prefijo gl representa la librería gl, y el sufijo 3f significa que la función toma tres argumentos en coma flotante. Todas las funciones OpenGL aceptan el siguiente formato:

<comando raíz><cuenta opcional de argumentos><tipo de argumentos opcional>

6.2 LA LIBRERIA AUX

La API OpenGL está dividida en tres librerías distintas:

Librería auxiliar o AUX (glaux.lib). Las declaraciones para esta librería se encuentran en el fichero glaux.h. Se trata de una librería de recursos que proporciona una estructura de trabajo independiente de la plataforma para invocar a las funciones de OpenGL. Todas sus funciones empiezan con el prefijo *aux*.

Funciones que definen actualmente OpenGL. Estan contenidas en la librería opengl32.dll y su cabecera es gl.h. Las funciones de esta librería tienen el prefijo *gl*.

Librería de utilidades (glu32.dll).Su cabecera es glu.h, y contiene funciones útiles para facilitar el trabajo, como el dibujo de esferas, discos y cilindros. Todas estas funciones tienen el prefijo *glu*.

Todas las funciones de las librerías opengl32.dll y glu32.dll están disponibles cuando se usa la librería AUX para la estructura de trabajo de su programa.

La librería AUX fue creada para facilitar el aprendizaje y escritura de programas en OpenGL sin distraerse con las particularidades de un entorno concreto como UNIX o Windows. Cuando la usamos, no escribimos el código final sino un escenario de fondo preliminar para probar nuestras ideas. Mediante el uso de la librería AUX para crear y gestionar la ventana y la interacción con el usuario, así como permitir a OpenGL que dibuje, es posible escribir programas que creen imágenes enormemente complejas. AUX también implementa funciones para permitir operaciones específicas del sistema, como el intercambio de buffers y la apertura de imágenes. Cuanto más use nuestro código estas funciones auxiliares, menos portable será.

Hay que tener en cuenta que OpenGl es independiente de la plataforma y por eso mismo no tiene ninguna función relacionada con la gestión de ventanas o de la pantalla o para las entradas desde teclado e interacción con el ratón.

Sin embargo, la librería AUX proporciona unas funciones rudimentarias para crear una ventana y leer la actividad del teclado y del ratón. También contiene funciones para el dibujo de algunos objetos 3D relativamente simples como esferas, cubos, toros, etc.

Hay que destacar que el proyecto que se ha desarrollado se ha utilizado el programa QtCreator para crear la ventana y poder interactuar con el usuario mediante el teclado y el ratón.

6.3 BUFFERS PARA LA INFORMACIÓN DE COLOR Y PROFUNDIDAD

buffers para la información de color y profundidad. OpenGL proporciona diversos tipos de buffers que están enlazados por el contexto gráfico de OpenGL:

- Buffer de color.
- Buffer de profundidad.
- Buffer de estarcido.
- Buffer de acumulación.

Cada buffer tiene propiedades específicas que van más allá que el simple doble buffer para animación y un buffer de ocultación para eliminar caras ocultas.

En OpenGL un buffer es esencialmente una matriz bidimensional de valores que se corresponden con un píxel en una ventana o en una imagen fuera de pantalla. Cada buffer tiene el mismo número de filas y columnas que el área cliente actual de la ventana, pero mantiene un tipo y rango de valores distinto.

Antes de usar OpenGL, se debe configurar el contexto de dispositivo hardware de la ventana (HDC) con los buffers y el modo de color que se requiere. La estructura PIXELFORMATDESCRIPTOR contiene esta información.

6.4 EL BUFFER DE COLOR

El buffer de color contiene información de color de los píxels. Cada píxel puede contener un índice de color o valores rojo/verde/azul/alfa que describen la apariencia de cada píxel. Los píxels RGBA se visualizan directamente en pantalla utilizando el color más próximo disponible.

La apariencia de los pixels de color con índice viene determinada por el valor asociado a este índice en una tabla de color RGB.

6.5 DOBLE BUFFER

El doble buffer proporciona un buffer de color adicional fuera de la pantalla que se usa a menudo para animación. Con el doble buffer se puede dibujar una escena fuera de la pantalla e intercambiarla rápidamente con la que está en pantalla, eliminando el parpadeo.

El doble buffer sólo afecta al buffer de color y no proporciona un segundo buffer de profundidad, acumulación o estarcido. Si se elige un formato de píxel con doble buffer, OpenGL selecciona el buffer oculto para dibujar. Se puede cambiar usando la función glDrawBuffer para especificar uno de los siguientes valores: (Tabla 9)

Buffer	Descripción
GL_FRONT	Dibuja sólo en el buffer de color frontal (visible).
GL_BACK	Dibuja sólo en el buffer de color trasero (oculto).
GL_FRONT_AND_BACK	Dibuja en ambos buffers de color.

(Tabla 9)

6.6 INTERCAMBIO DE BUFFERS

OpenGl soporta doble buffer, pero no hay ninguna función para intercambiar los buffers frontal y oculto. Cada sistema de ventanas con OpenGL soporta una función para hacerlo. Bajo windows, esta llamada es:

SwapBuffers(hdc);

donde hdc es el contexto de dispositivo de la ventana en la que estamos dibujando.

6.7 EL BUFFER DE PROFUNDIDAD

El buffer de profundidad mantiene valores de distancia para cada píxel. Cada valor representa la distancia al píxel desde la posición del observador, y se escala para quedar dentro del volumen de trabajo actual. Este buffer se utiliza normalmente para ejecutar la eliminación de caras ocultas, pero también puede usarse para otros efectos especiales, como realizar un corte en los objetos para ver la superficie interior.[4]

6.8 COMPARACIONES DE PROFUNDIDAD

Cuando se dibuja en una ventana OpenGL, la posición Z de cada píxel se compara con un valor del buffer de profundidad. Si el resultado de la comparación es true, se almacena el píxel en el buffer de profundidad junto con su profundidad. OpenGL establece ocho funciones de comparación: (Tabla 10)

Nombre	Función
GL_NEVER	Siempre false.
GL_LESS	True si Z de referencia<Z de profundidad.
GL_EQUAL	True si Z de referencia=Z de profundidad.
GL_LEQUAL	True si Z de referencia<=Z de profundidad.
GL_GREATER	True si Z de referencia>Z de profundidad.
GL_NOTEQUAL	True si Z de referencia!=Z de profundidad.
GL_GEQUAL	True si Z de referencia=>Z de profundidad.
GL_ALWAYS	Siempre true.

(Tabla 10)

La función por defecto es GL_LESS. Para cambiarla, se llama :

glDepthFunc(función);

Usando la función GL_LESS, los píxeles de un polígono se dibujan si su valor de profundidad es menor que el valor de profundidad situado en el buffer.

6.9 VALORES DE PROFUNDIDAD

En las comparaciones de profundidad GL_EQUAL y GL_NOTEQUAL, algunas veces es necesario alterar el rango de valores de profundidad utilizado, para reducir el número de valores posibles (manteniendo este número en el mínimo). Se ha de usar glDepthRange como sigue:

glDepthRange(cerca, lejos);

Los parámetros cerca y lejos son valores de coma flotante entre 0.0 y 1.0, que son los valores por defecto. Normalmente, cerca tiene un valor inferior a lejos, pero podemos invertir ésto para obtener efectos especiales (o utiliar las funciones GL_GREATER y GL_GEQUAL). Al reducir el rango de valores almacenados en el buffer de profundidad, no se modifica el volumen de trabajo, pero reducirá la precisión del buffer de profundidad y puede llevar a errores en la eliminación de caras ocultas.

Algunas comparaciones de profundidad necesitan un valor inicial de profundidad distinto. Por defecto, el buffer de profundidad se limpia con 1.0 con la función glClear. Para especificar un valor diferente se ha de usar la función glClearDepth:

glClearDepth(profundidad);

El parámetro profundidad es un valor de coma flotante entre 0.0 y 1.0, a menos que se haya definido un rango menor con glDepthRange. En general, usaremos un valor de 0.0 para las comparaciones GL_GREATER y GL_GEQUAL, y 1.0 para las comparaciones GL_LESS y GL_LEQUAL.

6.10 DIBUJAR FORMAS CON OPENGL

6.10.1 DIBUJO DE PUNTOS EN 3D

Cuando se comienza a dibujar en un ordenador, normalmente se trabaja con pixels: dibujar un punto en alguna parte y darle un color específico. Con OpenGL es significativamente distinto. No se trabaja con coordenadas físicas de pantalla, ni píxeles, sino con *coordenadas posicionales* en nuestro volumen de visualización. Este volumen se establece con una llamada a la función glOrtho().

```
 glMatrixMode (GL_MODELVIEW);
 glLoadIdentity();
}
```

Para especificar un punto de dibujo en la "paleta" 3D, se usa la función de OpenGL: glVertex(), sin duda la función más usada de toda la API. La siguiente línea de código especifica un punto en nuestro sistema de coordenadas localizado 50 unidades en la dirección del eje x, 50 unidades en la dirección del eje y, 0 unidades en el eje z:

glVertex(50.0f,50.0f,0.0f);

La definición geométrica de un vértice es no es un punto en el espacio, más bien el punto en el que dos líneas o curvas se intersectan. Esta es la esencia de las primitivas. Una primitiva es la interpretación de una lista de vértices en alguna forma dibujada en la pantalla. Hay diez primitivas en OpenGL, desde un simple punto hasta un polígono cerrado de cualquier número de caras. Se usa el comando glBegin para decirle a OpenGL que comienze a interpretar una lista de vértices como una primitiva particular. Entonces se termina la lista de vértices para esa primitiva con el comando glEnd.

```
glBegin(GL_POINTS);
glVertex3f(0.0f,0.0f,0.0f);
glVertex3f(50.0f,50.0f,50.0f);
glEnd();
```

El argumento GL_POINTS le dice a OpenGL que los vértices siguientes deben interpretarse como puntos. Podemos listar múltiples primitivas entre llamadas a glBegin y glEnd mientras que sean para el mismo tipo de primitiva.

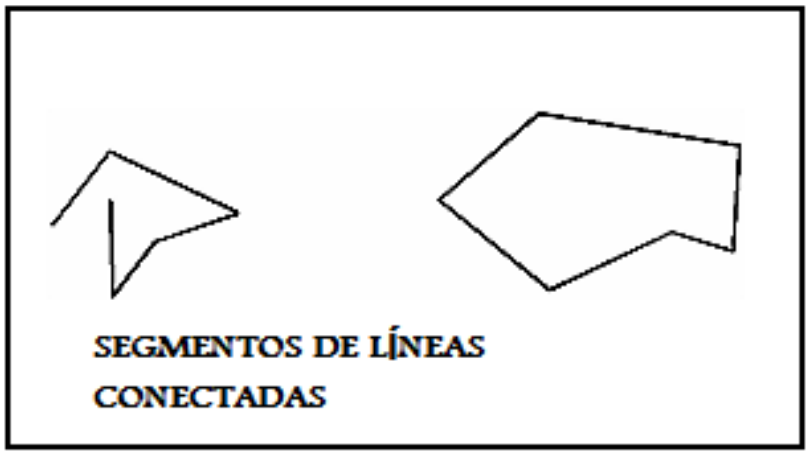
6.10.2 DIBUJO DE LINEAS EN 3D

La primitiva GL_POINTS es muy cómoda; por cada vértice especificado, dibuja un punto. El siguiente paso lógico es especificar dos vértices y dibujar una línea entre ellos. Esto es lo que hace la próxima primitiva, GL_LINES. El siguiente recorte de código dibuja una línea entre dos puntos (0,0,0) y (50,50,50):

```
glBegin(GL_LINES);
glVertex3f(0.0f,0.0f,0.0f);
glVertex3f(50.0f,50.0f,50.0f);
glEnd();
```

Para cada dos vértices, se dibuja una línea. Si especificamos un número impar de vértices, el último se ignora.

Las dos siguientes primitivas OpenGL se construyen sobre GL_LINES al permitirnos especificar una lista de vértices a través de los cuales se dibuja una línea. Cuando usamos GL_LINES_STRIP, se dibuja una línea desde un vértice al siguiente en un segmento contínuo. GL_LINE_LOOP es la última primitiva de líneas y es igual que la anterior, pero se dibuja una línea entre el último vértice especificado y el primero especificado. (FIGURA 68)



(FIGURA 68)

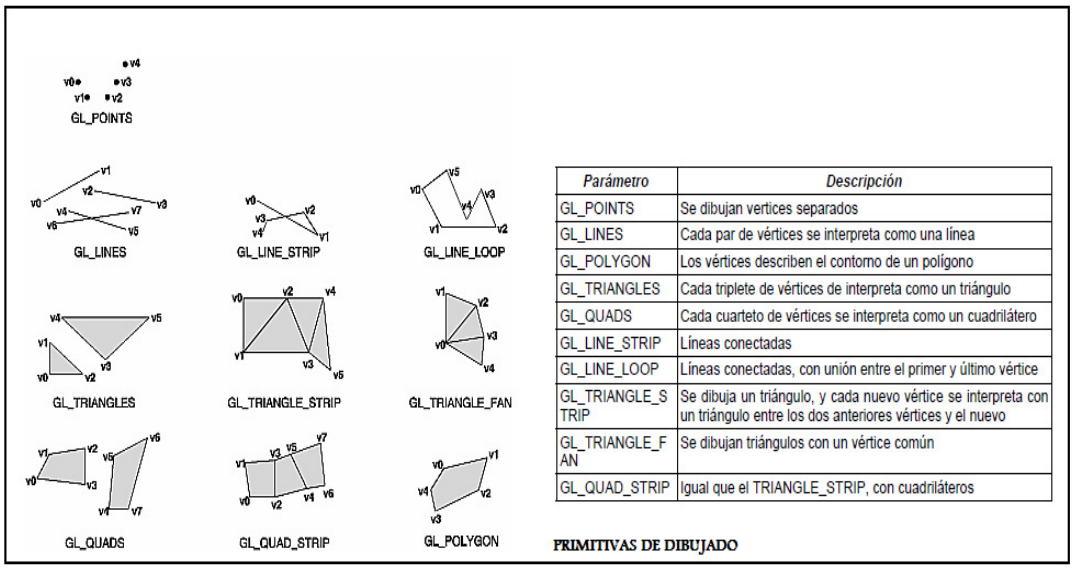
6.10.3 DIBUJO DE TRIANGULOS EN 3D

Con las primitivas vistas anteriormente se puede comenzar a dibujar cualquier cosa que se desee, pero estas formas no estarán rellenas de ningún color. Para dibujar una superficie sólida, necesitamos algo más que puntos y líneas; necesitamos polígonos, que son formas cerradas que pueden o no estar rellena con el color de dibujo actual, y es la base de toda composición de objetos sólidos con OpenGL.

El polígono más simple es el triángulo, que se construye mediante la primitiva GL_TRIANGLES, y lo hace conectando tres vértices. Para muchas superficies y formas, necesita dibujar muchos triángulos conectados. Se puede ahorrar mucho tiempo dibujando una tira de triángulos conectados con la primitiva GL_TRIANGLE_STRIP. Hay dos ventajas en usar una tira de triángulos en lugar de especificar cada triángulo separadamente. Primero, tras especificar los tres primeros vértices para el triángulo inicial, sólo necesitamos especificar un único punto para cada triángulo adicional. Esto ahorra mucho tiempo, al igual que espacio de datos. La segunda ventaja es que es una buena idea componer un objeto o superficie a base de triángulos en lugar de con alguna otra primitiva.

Además de tiras de triángulos, se puede usar GL_TRIANGLE_FAN para producir un grupo de triángulos conectados que se sitúan en torno a un punto central. El primer vértice V0, forma el origen de giro.

En la siguiente figura se muestra las distintas funciones que brinda OpenGL para crear las distintas figuras geométricas.(FIGURA 69)



Parámetro	*Descripción*
GL_POINTS	Se dibujan vertices separados
GL_LINES	Cada par de vértices se interpreta como una línea
GL_POLYGON	Los vértices describen el contorno de un polígono
GL_TRIANGLES	Cada triplete de vértices de interpreta como un triángulo
GL_QUADS	Cada cuarteto de vértices se interpreta como un cuadrilátero
GL_LINE_STRIP	Líneas conectadas
GL_LINE_LOOP	Líneas conectadas, con unión entre el primer y último vértice
GL_TRIANGLE_STRIP	Se dibuja un triángulo, y cada nuevo vértice se interpreta con un triángulo entre los dos anteriores vértices y el nuevo
GL_TRIANGLE_FAN	Se dibujan triángulos con un vértice común
GL_QUAD_STRIP	Igual que el TRIANGLE_STRIP, con cuadriláteros

PRIMITIVAS DE DIBUJADO

(FIGURA 69)

6.10.4 OTRAS PRIMITIVAS

Los triángulos son la primitiva preferida para la composición de objetos dado que la mayoría de hardware OpenGL acelera específicamente triángulos, pero no es la única primitiva disponible. Las restantes proporcionan una especificación rápida de cuadriláteros o tiras de cuadriláteros, al igual que un polígono de propósito general. La primitiva GL_QUADS dibuja un polígono de cuatro lados. De igual forma que podemos hacerlo con triángulos, podemos especificar una tira de cuadriláteros conectados con la primitiva GL_QUAD_STRIP.

La última primitiva OpenGL es GL_POLYGON (FIGURA 70), que puede emplearse para dibujar un polígono que tenga cualquier número de lados. Las diez primitivas se usan con glBegin/glEnd para dibujar formas poligonales de propósito general. Una forma es tan común que tiene una función especial en vez de ser una primitiva, esta forma es el rectángulo. La función glRect() proporciona un mecanismo sencillo y conveniente para especificar rectángulos sin tener que recurrir a GL_QUAD.

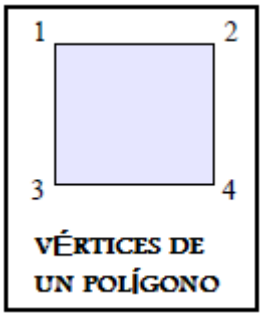
Cuando se usa muchos polígonos para construir una superficie compleja, se necesita recordar dos reglas importantes. La primera es que todos los polígonos deben ser planos. Esto es, todos los vértices del polígono deben estar en un mismo plano. El polígono no puede retorcerse ni doblarse en el espacio. La segunda regla es que los lados de los polígonos no deben intersecarse, y que los polígonos deben ser convexos. Un polígono se interseca a sí mismo si dos líneas cualquiera de su perímetro se cruzan. "Convexo" significa que en el polígono no hay huecos. Una prueba más rigurosa de un polígono convexo es dibujar algunas lineas sobre él. Si cualquier línea dada se introduce y deja el polígono mas d euna vez, el polígono no es convexo.

(FIGURA 70)

6.10.5 ORIENTACIÓN DE LAS CARAS EN OPENGL

Un polígono tiene dos caras, delantera y trasera. La manera de saber qué cara es la delantera y cual la trasera, es que, sí se mira la delantera, los vértices se habrán dibujado en orden antihorario.

Por ejemplo, tenemos la siguiente figura(FIGURA 71):



(FIGURA 71)

Si dibujamos los vertices en el orden 1,3,4,2, estaremos dibujando la cara delantera mirando hacia nosotros, pero si el orden es, por ejemplo, 1,2,4,3, estaremos mirando la cara trasera del polígono.

6.10.6 POLIGONOS COMPLEJOS

En OpenGL, un polígono complejo es aquél cóncavo o que tiene agujeros dentro de él. La primitiva GL_POLYGON de OpenGL sólo puede generar polígonos convexos simples. Un polígono es convexo si no hay ningún punto en una línea comprendida entre dos vértices cualquiera. Los polígonos cóncavos son polígonos no convexos que no tienen huecos en su interior. Los polígonos complejos tienen agujeros o están retorcidos.

6.10.7 COMPRENSION DE LAS TRANSFORMACIONES

Las *transformaciones* hacen posible la proyección de coordenadas 3D sobre una pantalla 2D. También nos permiten rotar los objetos, moverlos y cambiarlos de tamaño. Más que modificar nuestros objetos directamente, una transformación modifica el sistema de coordenadas.

Hay tres tipos de transformaciones que ocurren entre el instante en que especificamos nuestros vértices y el momento en que aparecen en pantalla: del observador, del modelado y de la proyección.

6.10.8 TRANSFORMACIONES DEL OBSERVADOR

Es la primera que se aplica y se usa para determinar el punto más ventajoso de la escena. Por defecto está en el origen (0,0,0) mirando en la dirección negativa del eje z (hacia dentro de la pantalla). La transformación del observador nos permite emplazar el punto de observación donde queramos, y mirando en cualquier dirección. Es la primera que se especifica porque todas las transformaciones poteriores tienen lugar basadas en el nuevo sistema de coordenadas modificado.

6.10.9 TRANSFORMACIONES DEL MODELO

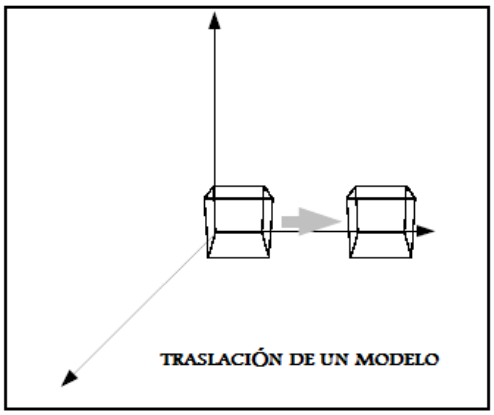
Se usan para manipular nuestro modelo y los objetos particulares que contiene. Esta transformación coloca los objetos en su lugar, los gira y los escala. La siguiente figura ilustra tres transformaciones que podemos realizar a nuestros objetos: translación, donde un objeto se desplaza a lo largo de unos ejes, rotación donde un objeto se gira sobre unos ejes, y escalado, donde se incrementan o disminuyen las dimensiones del objeto.

- glTranslate*: nos va a permitir trasladar un objeto en el espacio
- glRotate*: nos va a permitir rotar un objeto
- glScale*: nos va a permitir escalar un objeto
- glMultMatrix: multiplica la matriz de transformación actual por una matriz dada.

  Esto es algo muy utilizado en motores 3D

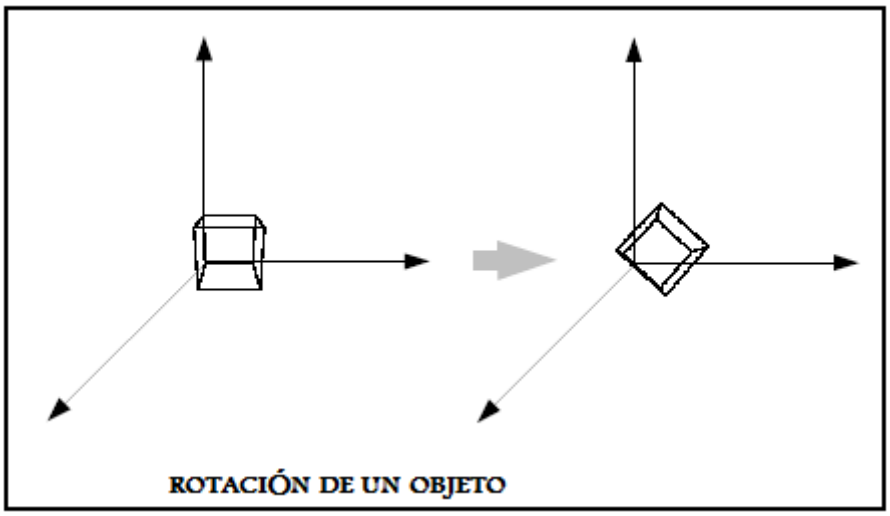
Es muy importante el orden en el que vayamos a realizar las transformaciones. No es lo mismo trasladar algo y después rotarlo, que primero rotarlo y luego trasladarlo.

Traslación: una traslación es un desplazamiento de un objeto en el espacio. Por ejemplo, movemos un objeto a una nueva posición desde la actual.(FIGURA 72)



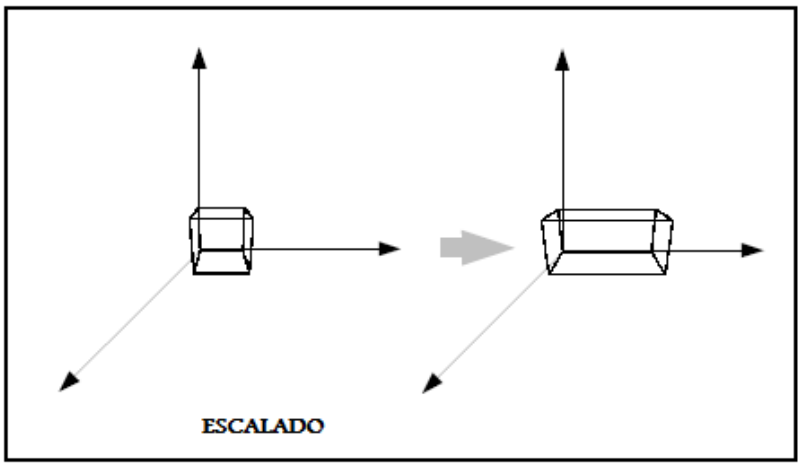
(FIGURA 72)

Rotación: como su nombre indica, un objeto rota alrededor de un eje que pasa por su "centro de giro".(FIGURA 73)



(FIGURA 73)

Escalado: Afecta al tamaño con que se visualiza el objeto por su transformación de escalado.(FIGURA 74)



(FIGURA 74)

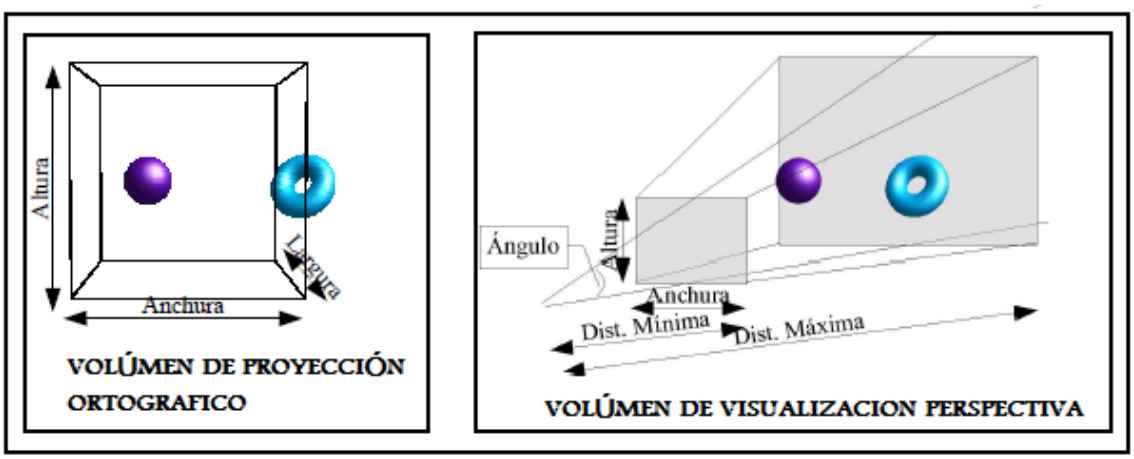
6.11 TRANSFORMACIONES DE LA PROYECCION

Se aplica a la orientación final del modelador. Esta proyección define el volumen de visualización y establece los planos de trabajo. Es decir, especifica cómo se traslada una escena finalizada a la imagen final en la pantalla. Ha de verse estos dos tipos: *ortográfica* y *en perspectiva*.

En la primera, todos los polígonos se dibujan en pantalla exactamente con las mismas dimensiones relativas especificadas. La proyección en perspectiva muestra los objetos más como deben aparecer en la vida real. Su característica más acusada es el empequeñecimiento, que hace que los objetos lejanos aparezcan más pequeños que los cercanos. Normalmente se ha de usar la proyección en perspectiva, a no ser que se necesite modelar objetos simples a los que no afecta ni la posición ni la distancia del observador, con los que usaremos proyección ortográfica.

void gluPerspective(GLdouble angulo, GLdouble aspecto, GLdouble zFrontal, GLdouble zPosterior);

Los parámetros son: un ángulo para el campo de visión en sentido vertical, la relación entre la altura y la anchura y la distancia entre los planos de trabajo frontal y posterior. La siguiente imagen (FIGURA 75) muestra la misma caja de antes pero con perspectiva:



(FIGURA 76)

6.12 MATRICES

Cada una de las transformaciones comentadas pueden realizarse multiplicando una matriz que contenga los vértices por una matriz que describa la transformación. Para que el tipo de transformaciones descrito tenga efecto, modificaremos dos matrices en particular: la matriz del modelador y la de la proyección. El camino que va de los datos en bruto de los vértices hasta las coordenadas de pantalla es bastante largo.

Primero, el vértice se convierte en una matriz 1x4 en la que los tres primeros valores son las coordenadas x,y,z. El cuarto número es el factor de escala w. Entonces se multiplica el vértice por la matriz del modelador, con lo que obtenemos las coordenadas oculares, que se multiplican por la matriz de proyección para obtener las coordenadas de trabajo. Esto elimina efectivamente todos los todos los datos fuera del volumen de visualización. Las coordenadas de trabajo se dividen entonces por la coordenada w para conseguir las coordenadas de dispositivo normalizadas. Finalmente, nuestra terna de coordenadas se mapea a un plano 2D mediante la transformación de la vista. esto también se representa con una matriz, que OpenGL definirá internamente dependiendo de los valores que especifiquemos en glViewport.

6.12.1 LA MATRIZ DEL MODELADOR

Es una matriz 4x4 que representa el sistema de coordenadas transformado que estamos usando para colocar y orientar nuestros objetos. Los vértices que proporcionamos a nuestras primitivas se emplean con una matriz de una sóla columna y se multiplican por la matriz del modelador para conseguir unas nuevas coordenadas oculares.

Traslación

Suponga que se quiere dibujar un cubo en el origen que midiese 10 uds de largo. Para mover el cubo 10 uds hacia arriba en el eje y antes de dibujarlo se puede multiplicar la matriz del modelador por una matriz que describiese una traslación de 10 uds hacia arriba en el eje y, y entonces hacer el dibujo.
En la práctica, ésto no es necesario. Hay de una función de alto nivel que hace ésto :

void glTranslatef (GLfloat x, GLfloat y, GLfloat z);

Esta función toma como parámetros la traslación que hay que realizar en las direcciones x,y,z. Entonces construye una matriz apropiada y hace la multiplicación. Simplemente:

```
//Traslada 10 uds hacia arriba en el eje y
glTranslatef (0.0f,10.0f,0.0f);
//Ahora habría que Dibujar el cubo
```

Rotación

Para rotar un objeto sobre uno de los tres ejes, se tendrá que haber ideado una matriz de rotación para multiplicarla por la matriz del modelador. De nuevo, se ha de usar una función de alto nivel:

glRotatef (GLfloat angulo,GLfloat x, GLfloat y, GLfloat z);

Aquí se esta ejecutando una rotación sobre el vector descrito por x,y,z. El ángulo de rotación se mide en grados en sentido horario negativo y está especificado por el argumento ángulo.

Escalado

Una transformación de escala incrementa el tamaño del objeto expandiendo todos los vértices a lo largo de los tres ejes por los factores especificados. La función:

glScalef(GLfloat x, GLfloat y, GLfloat z);

Múltiplica los valores x,y,z por los factores de escala especificados. El escalado no tiene por qué ser uniforme. Se puede usar para estirar o encoger objetos. Por ejemplo, el siguiente código produce un cubo que es el doble de largo en los ejes x,z pero sigue igual en el eje y.

```
  //Ejecuta la transformación de escala
glScale (2.0f,1.0f,2.0f);
//Dibuja el cubo
auxWireCube(10.0f);
```

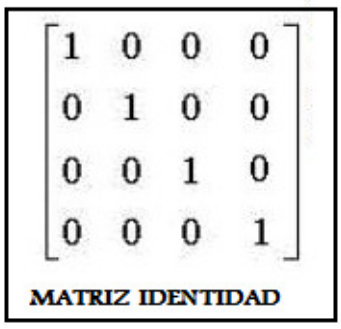
6.12.2 LA MATRIZ DE IDENTIDAD

Los efectos de las funciones antes descritas son acumulativos. Cada vez que se invoca una, la matriz apropiada se construye y multiplica por la matriz del modelador. La nueva matriz se convierte en la matriz del modelador actual, que es entonces multiplicada por la siguiente transformación, y sigue. Si no se quiere que ocurra, se puede reiniciar la matriz del modelador a un estado conocido, en este caso, centrada en el origen del sistema de coordenadas oculares.

Esto se consigue abriendo la matriz del modelador con la matriz de identidad. Esta matriz especifica que no ocurre ninguna transformación. Tiene todo ceros excepto la diagonal principal, que tiene unos. Las dos siguientes líneas abren la matriz de identidad en la matriz del modelador:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

La primera línea especifica que la actual matriz de operaciones es la matriz del modelador. La segunda línea abre la matriz actual con la matriz identidad.(FIGURA 77)

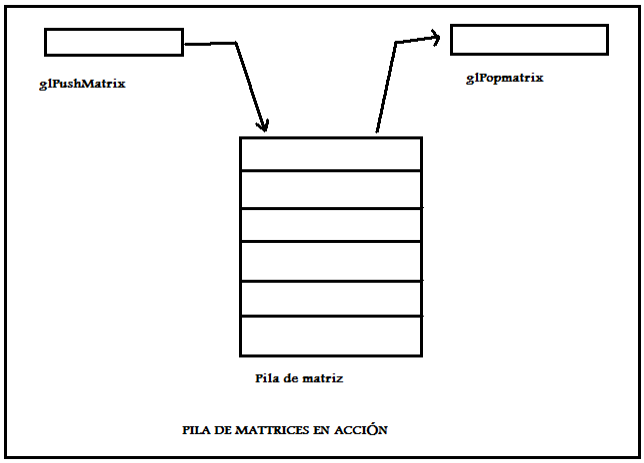


(FIGURA 77)

6.12.3 LAS PILAS DE MATRICES

No siempre es deseable reiniciar la matriz del modelador con la identidad antes de colocar cada objeto. A menudo se quiere almacenar el estado actual de transformación y entonces recuperarlo después de haber colocado varios objetos.

Para facilitar esto, OpenGL mantiene una pila de matrices para las matrices del modelador y de la proyección. Una pila de matrices trabaja de igual forma que la pila ordinaria de un programa.(FIGURA 78)



(FIGURA 78)

EJEMPLO

En el siguiente fragmento de código se mostrará como se utiliza la pila :

```
// limpia la ventana con el color actual de borrado
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// reinicia la matriz del modelador
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// Almacena la transformación de la vista
        glPushMatrix();
// Rotado por el ángulo de revolución
        glRotatef(XROT, 0.0f, 1.0f, 0.0f);
// Se traslada del origen a la distancia orbital
        glTranslatef(90.0f, 0.0f, 0.0f);
// Dibuja la figura
    glBegin(GL_POLYGON);
    glColor3f (0.0f,0.0f,1.0f);
    glVertex3f(0.0, 1.0, 0.0);//1
    glVertex3f(0.0, 0.0, 0.0);//2
    glVertex3f(1.0, 0.0, 0.0);//3
    glVertex3f(1.0, 1.0, 0.0);//4
// Reinicia la transformación de la vista
    glPopMatrix();
```

CAPÍTULO 7

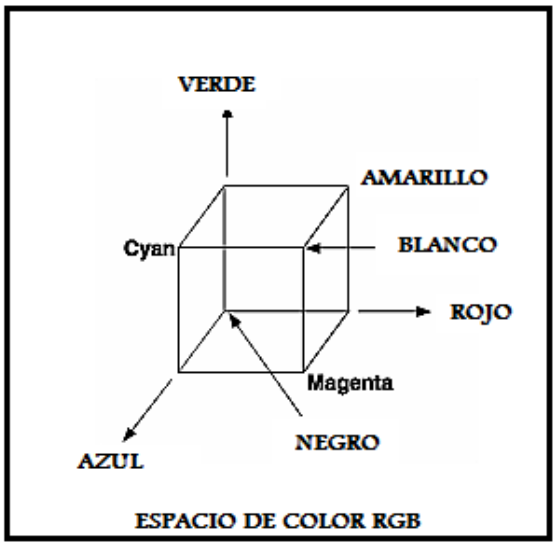
INTRODUCCIÓN: COLOR, MATERIAL

En este capítulo se hablará del color dado que esta característica dependerá mucho la apreciación que tenga el diseño que se vaya a desarrollar. Como se ha visto en el capítulo primero, por lo tanto de ahí la importancia del mismo.

El material también una parte fundamental delos diseños en tres dimensiones pero no es parte de nuestra tesis, por lo tanto solo se hará mención del mismo.

7.0 COLOR, MATERIAL

Dado que el color se especifica con tres valores positivos, se puede modelar los colores disponibles como un volumen al que se llamará el espacio de color RGB. La siguiente **figura** muestra el aspecto que tiene este espacio en el origen con el rojo, verde y azul en los ejes. En el origen, las intensidades relativas de todas las componentes es cero, y el color resultante es el negro. El extremo del cubo está situado en 255 unidades en cada cara, donde se encuentra el blanco. La saturación total (255) desde el origen a lo largo de cada eje dejaría los colores puros de rojo, verde y azul respectivamente.(F**IGURA 79**)



(FIGURA 79)

Este cubo de color contiene por tanto todos los colores posibles, tanto en la superficie como en el interior. Por ejemplo, todos los grados de grises se encuentran a lo largo de la diagonal entre las esquinas (0,0,0) y (255,255,255).

7.1 SELECCION DEL COLOR DE DIBUJO

Analizndo brevemente la función glColor(). La sintaxis es como sigue:

void glColor<x><t>(rojo, verde, azul, alfa);

En el nombre de la función, <x> representa el número de argumentos; debe ser 3 para argumentos de rojo, verde y azul, ó 4 para incluir la componente alfa, que especifica la transparencia. La <t> en el nombre, especifica el tipo de datos de los argumentos. La mayoría de los programas usan glColor3f() y especificarán 0.0 para ninguna intensidad, y 1.0 para toda intensidad. Sin embargo, si estamos acostumbrados a trabajar con colores en Windows, podemos usar la versión glColor3ub(), que toma tres bytes sin signo, entre 0 y 255, para las intensidades de las componentes roja, verde y azul. Es como usar la macro RGB de Windows:

```
glColor3ub(0,255,128) = RGB(0,255,128)
```

7.2 SOMBRA

La función glColor(), selecciona el color de dibujo para todos los vértices dibujados tras el comando. Un punto sólo tiene un vértice, y cualquiera que sea el color que se especifique para ese vértice resultará en el color de ese punto. Una línea, sin embargo, tiene dos vértices, y cada uno puede tener definido un color diferente. El color de la línea depende del *modelo de sombreado*. El sombreado se define como la transición suave entre un color y el siguiente. El siguiente código dibuja un triángulo degradado con esquinas roja, verde y azul:

```
// Activa el sombreado suave
glShadeModel(GL_SMOOTH);

// Dibuja el triángulo
glBegin(GL_TRIANGLES);
        // Vértice rojo
        glColor3ub((GLubyte)255,(GLubyte)0,(GLubyte)0);
        glVertex3f(0.0f,200.0f,0.0f);

        // Verde en la esquina inferior derecha
        glColor3ub((GLubyte)0,(GLubyte)255,(GLubyte)0);
        glVertex3f(200.0f,-70.0f,0.0f);
        // Azul en la esquina inferior izquierda
        glColor3ub((GLubyte)0,(GLubyte)0,(GLubyte)255);
        glVertex3f(-200.0f, -70.0f, 0.0f);
glEnd();
```

La primera línea del listado anterior selecciona el modelo de sombreado que OpenGL usa para realizar el degradado. Este es el modelo de sombreado por defecto. El otro modelo que puede especificarse con glShadeModel() es GL_FLAT para un *degradado plano*, que significa que no se realiza ningún cálculo de transición en el interior de las primitivas. Generalmente, con el degradado plano, el color interior de las primitivas es el del último vértice especificado, excepto con la primitiva GL_POLYGON, en cuyo caso el color es el del primer vértice.(FIGURA 80)

(FIGURA 80)

7.3 EL COLOR EN EL MUNDO REAL:

OpenGL hace un trabajo muy bueno de aproximación al mundo real en términos de condiciones de iluminación. A menos que un objeto emita su propia luz, está iluminado por tres tipos de luz diferentes: ambiente, difusa y especular.

7.4 MATERIALES EN EL MUNDO REAL

Cuando se utiliza, la iluminación no se describe los polígonos como que tengan un color particular, si no más bien que están hechos de *materiales* que tienen ciertas propiedades reflectivas. Se debe especificar las propiedades reflectivas de un material para las fuentes de luz ambiental, difusa y especular. Por ejemplo, un material puede ser brillante y reflejar muy bien la luz especular, al tiempo que absorbe la mayor parte de la luz ambiente y difusa. Otra propiedad que hay que especificar es la propiedad de emisión de los objetos que emiten su propia luz, como cerillas, etc.

CAPÍTULO 8

INTRODUCCIÓN: TEXTURAS

Una más de las herramientas que proporciona OpenGL es la opción de rellenar un polígono convexo con textura o con color como se vio en el capítulo anterior, o con combinación de ambas. El uso de estas herramientas dependerá del nivel de complejidad y realismo que deseé dar el diseñador a sus gráficos.

8.0 BASES DEL MAPEADO CON TEXTURAS

Toda textura es una imagen de alguna clase. Hay dos tipos:

Textura 1D.Es una imagen con anchura pero sin altura, o viceversa; tienen un único pixel de ancho o de alto.

Textura 2D.Es una imagen con más de un pixel de alto o ancho y se abre generalmente con un fichero .BMP. Se usan comunmente para reemplazar complejas geometrías de superficie en edificios, árboles y demás.

Todas estas texturas se componen de valores de color RGB y pueden incluir valores alfa (de transparencia). Naturalmente, debemos definir una imagen para textura antes de que podamos dibujar polígonos texturados en OpenGL. Las texturas siguen las mismas reglas de almacenamiento que Los mapas de bits.

8.1 DEFINICION DE TEXTURAS 1D

OpenGL proporciona una única función para definir texturas 1D, glTextImage1D, que acepta ocho argumentos:

*void glTextImage1D(GLenum objetivo, GLint nivel, GLint componentes, GLsizei ancho, GLint borde, GLenum formato, GLenum tipo, const GLvoid *pixels);*

El argumento objetivo especifica qué textura hay que definir; este argumento debe ser GL_TEXTURE_1D. El argumento nivel indica el nivel de detalle, y normalmente es 0. El argumento componentes especifica el número de valores de color usados en cada pixel. Para las texturas RGB y RGBA se usan 3 y 4, respectivamente. Ancho y borde especifican el tamaño de la textura. El valor borde controla el número de pixels de bordes que OpenGL debe esperar y puede valer 0, 1 ó 2. El valor ancho especifica la anchura de la textura original (sin borde) y debe ser una potencia de 2. El argumento formato indica el tipo de valores de color esperados, GL_COLOR_INDEX, GL_LUMINANCE, GL_RGBA.

8.2 DEFINICION DE TEXTURAS 2D

Para definir una textura 2D en OpenGL, llamamos a glTextImage2D, que toma un argumento de altura aparte de los que usa glTextImage1D.

*void glTextImage2D(GLenum objetivo, GLint nivel, GLint componentes, GLsizei ancho, GLsizei alto, GLint borde, GLenum formato, GLenum tipo, const GLvoid *pixels);*

Los argumentos ancho y alto deben ser potencia de 2.

8.3 DIBUJO DE POLIGONOS CON TEXTURAS

Una vez que se ha definido una textura, luego hay que activar el texturado. Para activar las texturas 1D usando las siguientes funciones que provee opengl:

glDisable(GL_TEXTURE_2D);
glEnable(GL_TEXTURE_1D);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

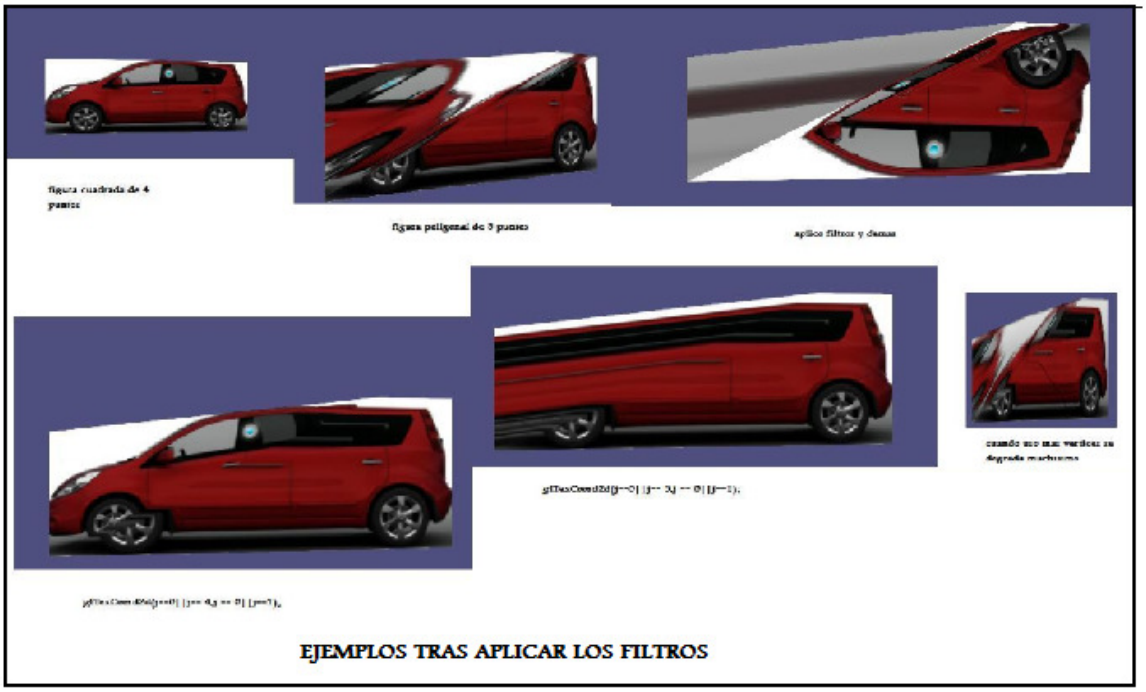
La llamada a glEnable activa las texturas 1D. La función glTexEnvi selecciona el modo de texturas "calco", que significa que las imágenes se superponen directamente en los polígonos. Otros modos son(Tabla 11):

Modo	Descripción
GL_MODULATE	Los pixels de la textura filtran los pixels de color existentes en la pantalla
GL_DECAL	Los pixels de la textura reemplazan los pixels de color existentes en la pantalla
GL_BLEND	Los pixels de la textura filtran los pixels de color existentes y se combinan con un color constante

(Tabla 11)

El modo de textura GL_MODULATE multiplica el color de la textura por el de la pantalla. Para texturas de una componente (luminancia), ésto se traduce en un filtro de brillo que variará el brillo de la pantalla basándose en la textura. Para texturas de tres componentes (RGB), podemos generar efectos de "filtro de lentes coloreadas".

A diferencia del texturado GL_MODULATE, el texturado GL_BLEND permite mezclar un color constante en la escena basándonos en la textura. Una vez que hemos definido el modo de textura que hay que usar, podemos proceder a dibujar en nuestros polígonos. El siguiente ejemplo (FIGURA 81) muestra como quedaría una textura al aplicarse sobre una figura plana contenida en el mismo plano, tras aplicarse una serie de filtros



(FIGURA 81)

Para (FIGURA 81) del ejemplo que contiene la textura se han aplicado los filtros deseados de magnificación y minimización, GL_LINEAR. EL filtro de minimización se usa cuando el polígono dibujado es más pequeño que la textura. El filtro de magnificación se usa cuando el polígono es más grande que la textura. Designando el filtro GL_LINEAR, le decimos a OpenGL que interpole linealmente los valores de color de la textura antes de dibujar. Los filtros son los siguientes:

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

Otros filtros que se puede usar con GL_TEXTURE_MIN_FILTER son (Tabla 12):

Filtro	Descripción
GL_NEAREST	Filtraje de vecino más próximo.
GL_LINEAR	Interpolación lineal.
GL_NEAREST_MIPMAP_LINEAR	Filtraje de vecino más cecano con textura multimapa.
GL_LINEAR_MIPMAP_LINEAR	Interpolación lineal de textura multimapa interpolada.
GL_LINEAR_MIPMAP_NEAREST	Interpolación lineal de textura multimapa.

(Tabla 12)

El filtrado GL_NEAREST toma el píxel más cercano en la textura, en lugar de interpolar entre pixels.

8.4 TEXTURAS MULTIMAPA

Hasta ahora, se ha tratado exclusivamente con texturas monomapa. Esto es, cuando se quiere dibujar un polígono texturado, éste se pintaba con una única imagen 1D o 2D. Esto está bien para algunas escenas, pero las secuencias animadas necesitan a menudo diferentes niveles de detalle dependiendo de la distancia del observador. Por ejemplo, cuando caminamos por una habitación virtual, podemos querer una vista de alta resolución de una foto de una imagen cercana, y sólo un bosquejo desde lejos.

OpenGl soporta texturas con múltiples imágenes, lo que se llama texturas *multimapa*. El multimapa selecciona la textura más cercana a la resolución de pantalla de un polígono. Las texturas multimapa se definen proporcionando un parámetro de nivel específico para cada mapa. Para la textura del ejemplo previo usaríamos lo siguiente:

Cuando se quiere dibujar polígonos con una textura multimapa, se debe usar uno de los filtros de minimización de la siguiente tabla(Tabla 13):

Filtro	Descripción
GL_NEAREST_MIPMAP_NEAREST	Utiliza la imagen más cercana a la resolución de la pantalla. Usaremos el filtro GL_NEAREST cuando usemos este mapa
GL_NEAREST_MIPMAP_LINEAR	Utiliza la imagen más cercana a la resolución de la pantalla. Usaremos el filtro GL_LINEAR cuando usemos este mapa
GL_LINEAR_MIPMAP_NEAREST	Interpola linealmente entre las dos imágenes más cercanas a la resolución de la pantalla. Usaremos el filtro GL_NEAREST cuando utilicemos este mapa.

GL_LINEAR_MIPMAP_LINEAR	Interpola linealmente entre las dos imágenes más cercanas a la resolución de la pantalla. Usaremos el filtro GL_LINEAR cuando utilicemos este mapa.

(Tabla 13)

Los filtros GL_LINEAR_MIPMAP_NEAREST y GL_LINEAR_MIPMAP_LINEAR pueden resultar muy costosos en términos de potencia de ejecución. GL_NEAREST_MIPMAP_NEAREST es muy parecido a GL_NEAREST en ejecución, pero generalmente produce resultados mucho mejores. Las imágenes multimapa se eligen comparando el tamaño de los polígonos a medida que se dibujan en la pantalla, con los tamaños de las texturas multimapa.

En la siguiente (FIGURA 82) se puede apreciar un ejemplo de varias texturas aplicadas en un mismo objeto.

(FIGURA 82)

8.5 FUNCIONES DE ESTADO BASICAS

El *mecanismo de estados* es una de las cosas que hacen que OpenGL sea tan rápido y eficiente dibujando gráficos 3D. Este estado se encuentra agrupado en categorías diferentes como color, iluminación, texturas, etc. Cada contexto de generación que creamos tiene su propio estado de generación específico para una ventana o mapa de bits fuera de la pantalla.

Las dos funciones de OpenGL que activan y desactivan las funciones de generación se llaman glEnable y glDisable. A estas funciones se les pasa una simple constante enumerada, como GL_DEPTH_TEST, como sigue:

*glEnable(GL_DEPTH_TEST); /*Activa la verificación del buffer de profundidad*/*

*glDisable(GL_DEPTH_TEST); /*Desactiva la verificación del buffer de profundidad*/*

8.6 GRABAR Y RECUPERAR ESTADOS

De la misma forma que OpenGL mantiene una pila de matrices de proyección, del modelador y de texturas, tiene una pila para el estado de generación actual. A diferencia de una pila de matrices, la pila de estado da mucho más control sobre lo que se quiere colocar (*push*) o quitar (*pop*) de la pila.

Las funciones OpenGL para almacenar y recuperar atributos del estado de generación son glPushAttrib y glPopAttrib. La función glPushAttrib trabaja de forma muy parecida a glPushMatrix, con la excepción de que se puede seleccionar los valores de estado que se quiera colocar en la pila. Para almacenar el estado de generación completo, se llama a:

glPushAttrib(GL_ALL_ATTRIB_BITS);

8.6.1 DIBUJAR ESTADOS

OpenGL tiene un gran número de estados asociados a las acciones de dibujo para las primitivas glBegin/glEnd básicas. La mayoría se almacenan con una llamada a glPushAttrib(GL_CURRENT_BIT|GL_LINE_BIT). Ver la siguiente tabla(Tabla 14):

Variable de estado	Descripción
GL_ALPHA_TEST	Efectúa la verificación de valores alfa.
GL_BLEND	Ejecuta las operaciones de mezcla de pixels.
GL_CLIP_PLANx	Operaciones de dibujo fuera del plano de trabajo especificado.
GL_CULL_FACE	Oculta los polígonos frontales o traseros.
GL_DITHER	Valores de color de fusión.
GL_LINE_SMOOTH	Líneas de antiescalonado.
GL_LINE_STIPPLE	Aplica un patrón de bits a las líneas.
GL_LOGIC_OP	Efectúa operaciones lógicas sobre los pixels al dibujarlos.
GL_POINT_SMOOTH	Puntos con antiescalonado.
GL_POLYGON_SMOOTH	Polígonos con antiescalonado.
GL_POLYGON_STIPPLE	Aplica un patrón de bits a los polígonos.
GL_SCISSOR_TEST	Limita el dibujo en las áreas exteriores a la región glScissor.

(Tabla 14)

Estados del buffer de dibujo. Un fallo muy común es olvidar activar la verificación de profundidad, y sin ella, no se ejecuta la iluminación de caras ocultas con el buffer de profundidad. Una llamada a glPushAtrrib con GL_DEPTH_BUFFER_BIT sirve para tomar la precaución de almacenar el estado GL_DEPTH_TEST.

Estados de textura. Para almacenar los parámetros actuales de texturizado, se llama a glEnable con GL_TEXTURE_BIT y GL_EVAL_BIT. Cuando se activa la textura, hay que asegurarse de que sólo se activa un modo de texturizado. La especificación OpenGL establece que el texturizado 2D prevalece sobre el 1D, pero algunas implementaciones no cumplen eso. La siguiente tabla (Tabla 15) recoge las variables disponibles:

Variable de estado	Descripción
GL_MAP1_TEXTURE_COORD_1	La coordenada de textura s se generará con llamadas

	a glEvalPoint1, glEvalMesh1 y glevalCoord1.
GL_MAP1_TEXTURE_COORD_2	Las coordenadas de textura s y t se generarán con llamadas a glEvalPoint1, glEvalMesh1 y glevalCoord1.
GL_MAP1_TEXTURE_COORD_3	Las coordenadas de textura s, t y r se generarán con llamadas a glEvalPoint1, glEvalMesh1 y glevalCoord1.
GL_MAP1_TEXTURE_COORD_4	Las coordenadas de textura s, t, r y q se generarán con llamadas a glEvalPoint1, glEvalMesh1 y glevalCoord1.
GL_MAP2_TEXTURE_COORD_1	La coordenada de textura s se generará con llamadas a glEvalPoint2, glEvalMesh2 y glevalCoord2.
GL_MAP2_TEXTURE_COORD_2	Las coordenadas de textura s y r se generarán con llamadas a glEvalPoint2, glEvalMesh2 y glevalCoord2.
GL_MAP2_TEXTURE_COORD_3	Las coordenadas de textura s, r y t se generarán con llamadas a glEvalPoint2, glEvalMesh2 y glevalCoord2
GL_MAP2_TEXTURE_COORD_4	Las coordenadas de textura s, r, t y q se generarán con llamadas a glEvalPoint2, glEvalMesh2 y glevalCoord2
GL_TEXTURE_1D	Activa el texturado 1D a menos que esté activo el texturado 2D
GL_TEXTURE_2D	Activa el texturado 2D
GL_TEXTURE_GEN_Q	Genera automáticamente la coordenada q a partir de llamadas a glVertex.
GL_TEXTURE_GEN_R	Genera automáticamente la coordenada r a partir de llamadas a glVertex.
GL_TEXTURE_GEN_S	Genera automáticamente la coordenada s a partir de llamadas a glVertex.
GL_TEXTURE_GEN_T	Genera automáticamente la coordenada t a partir de llamadas a glVertex.

(Tabla 15)

Estados de píxel. Los modos de transferencia, almacenamiento y mapeado de píxeles son probablemente los aspectos menos conocidos y optimizados de OpenGL. Los almacenamos con una llamada a glPushAttrib(GL_PIXEL_BIT). No hay estados glEnable para estos modos.

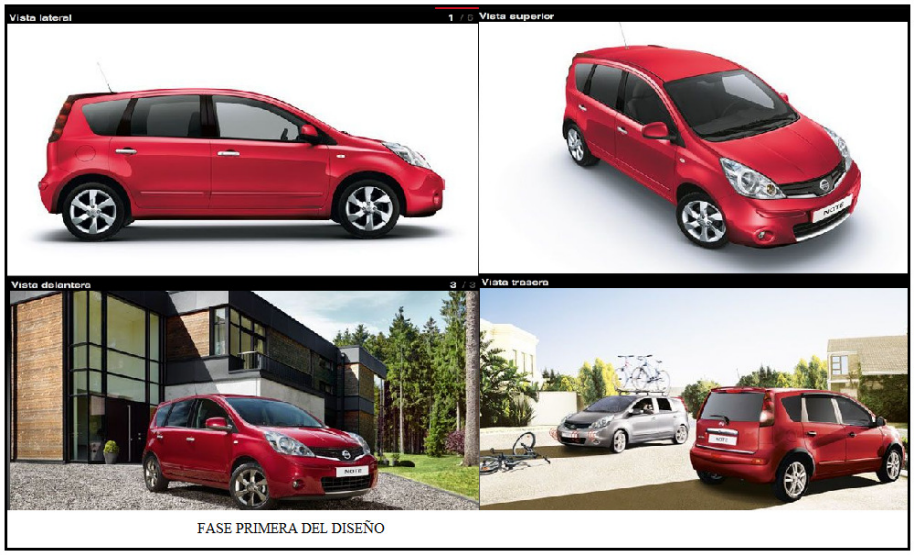
Capítulo 9

Diseños desarrollados

Es en este capítulo donde se mostrarán imágenes de los diseños desarrollados tras lograr la integración de la biblioteca gráfica OpenGL en QtCreator .

9.0 FASE PRIMERA DEL DISEÑO

Esta primera fase consiste en hacerse una idea del diseño. Después se ha de obtener cotas, alturas alejamientos y de ahí pasar dichas medidas al sistema de coordenadas que se desea, que este caso es el de tres dimensiones.**(FIGURA 83)**

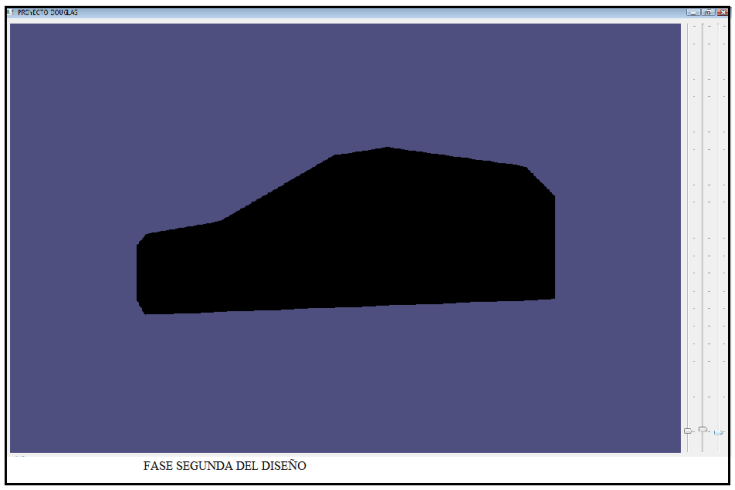


FASE PRIMERA DEL DISEÑO

(FIGURA 83)

9.1 FASE SEGUNDA

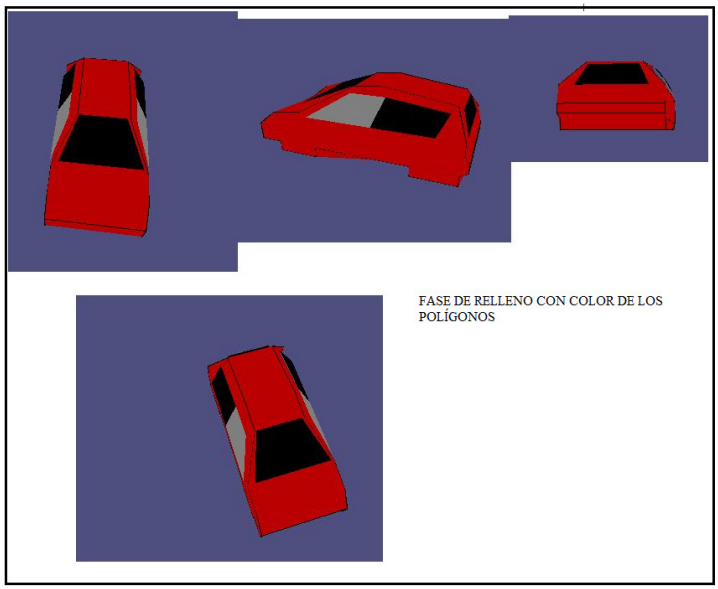
En esta fase del diseño se puede ver en la figura (FIGURA 84) como los vértices que se han introducido mediante las funciones vertex(x , y, z) , forman un polígono de n lados. Mediante el uso de la función glBegin(GL_POLYGON).


FASE SEGUNDA DEL DISEÑO

(FIGURA 84)

9.2 TERCERA FASE

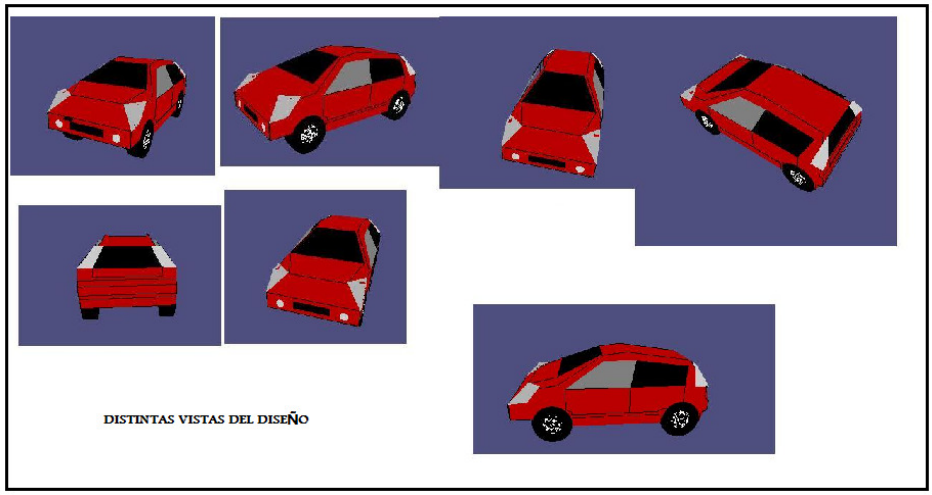
En esta la ventana (FIGURA 85) se puede distinguir mejor el diseño del vehículo, dado que se ha añadido diferentes elementos como son la ventanas y lunas delanteras y traseras.



(FIGURA 85)

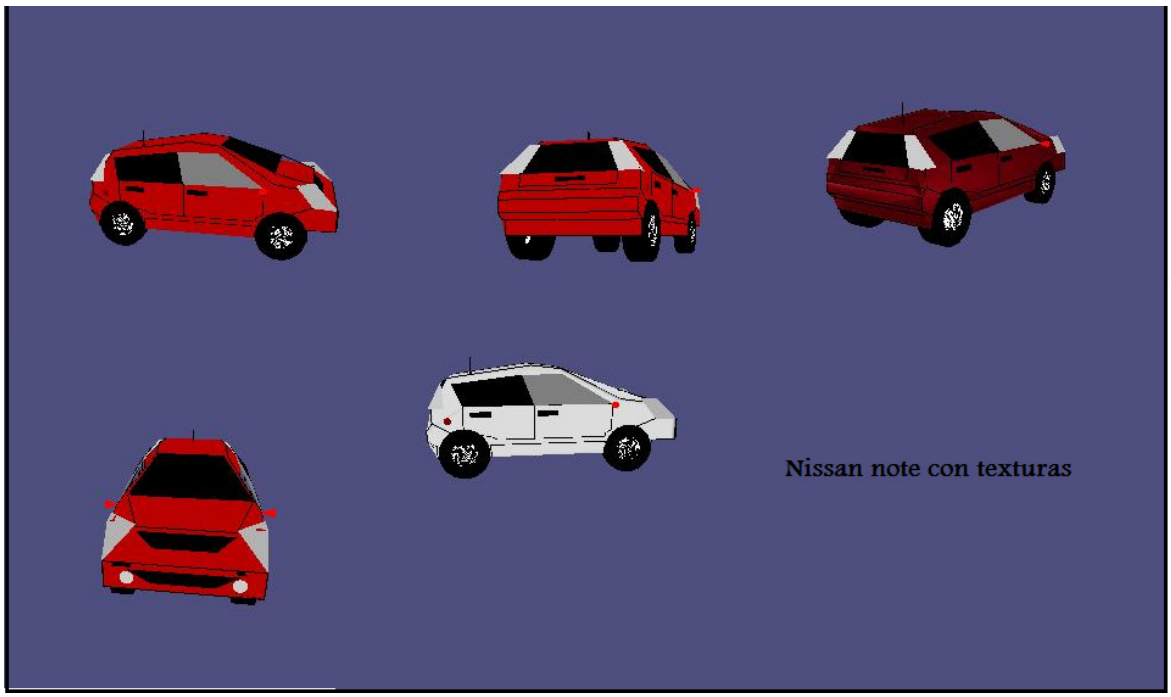
9.2 PRODUCTO FINAL

En la siguiente figura **(FIGURA 85)** se muestra el producto final para el diseño del vehículo Nissan "Note" tras haber usado las distintas herramientas de la biblioteca gráfica OpenGL.



(FIGURA 85)

9.4 POSIBLES MEJORAS



(FIGURA 86)

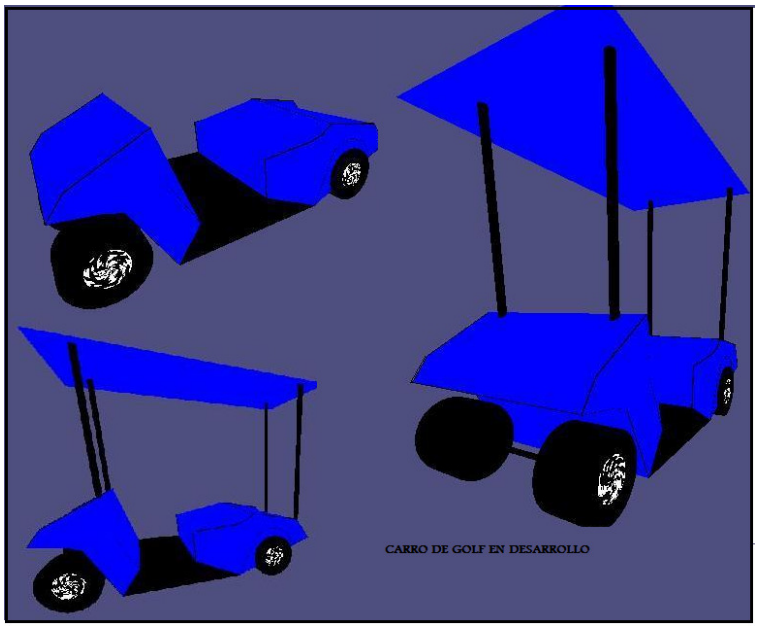
En el diseño de este vehículo se quiere mostrar el resultado final tras aplicar texturas. **(FIGURA 86)**

9.5 DISEÑO DEL CARRITO DE GOLF

En esta imagen **(FIGURA 87)** se muestra un carrito de golf real.

(FIGURA 87)

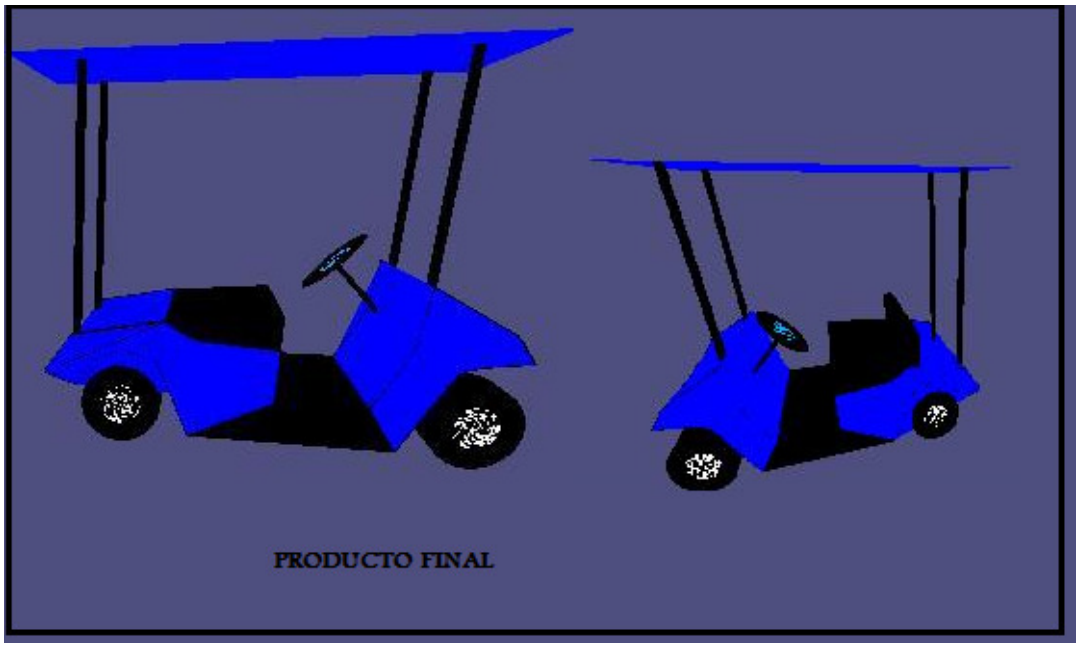
9.5.1 CARRITO DE GOLF EN PROCESO DE DESARROLLO



(FIGURA 88)

En la figura **(FIGURA 88)** se observa como se va desarrollando el diseño del vehículo

9.5.2 PRODUCTO FINAL



(FIGURA 89)

En la figura **(FIGURA 89)** se muestra el producto final del diseño.

Capítulo 10

Presupuesto

El presupuesto de este proyecto esta basado sobre todo en el número de horas que se ha tardado en la fase creativa del diseño, como es la adquisición de coordenadas, el pasar dichas coordenadas al código de C++, además de las infinitas simulaciones que se han tenido que realizar para llegar a un resultado aceptable.

Otra muy importante para el cálculo del presupuesto ha sido la integración de OpenGL en Qtcreaor.

PROYECTO FIN DE CARRERA
REALIZADO POR : D. DOUGLAS GUZMÁN CENTANARO
DIRIGIDO POR : D.ARTURO DE LA ESCALERA
ESPECIALIDAD : ELECTRÓNICA
SECCIÓN : AUTOMÁTICA
LEGANES A 16 DE NOVIEMBRE DEL 2011

PRESUPUESTO

CONSIDERACIONES PREVIAS

El presente presupuesto hace referencia a un proyecto del diseño de dos gráficos bajo la integración de una biblioteca gráfica en un programa gratuito para una aplicación concreta. Ha de comprenderse que en la materialización de este tipo de proyectos se utiliza muy pocos materiales únicamente se utiliza el ingenio y destreza de un programador, un ordenador y es por ello que el siguiente presupuesto solo se hace referencia al coste de las horas empleadas en su realización y a los posibles costes indirectos. Así mismo se ha añadido, por ser una parte fundamental e inseparable del programa, el coste de realización del manual del usuario.

A priori el resultado del presupuesto puede resultar excesivo, no obstante se tendrá en cuenta que la ejecución del programa no se ha realizado con vistas a una posible comercialización que diera lugar a su amortizacíon y rentabilización, se hace única y exclusivamente bajo el pedido de un cliente, en este caso el Departamento de Ingeniería en sistemas y automática de la Universidad Carlos III, con las especificaciones, características y prestaciones que este impuso. Se trata, por lo tanto, de un programa diseño original. realizado a medida que responde a unas necesidades concretas de un determinado cliente.

El presupuesto se va a dividir en costes directos e indirectos, desglosándose estos a su vez en otras partidas (tiempos para la programación de cada una de las partes que componen todo el conjunto).

COSTES DIRECTOS

Determinación del coste/hora por categorías

Para la determinación del coste/hora tendremos en cuenta los siguientes parámetros:

-Se precisa la intervención de un experto programador, que será el encargado de la realización del programa y de su correspondiente manual del usuario.
-No será necesaria la intervención de un analista de sistema: el conjunto ha de funcionar de forma correcta.
-No será necesaria la intervención de ningún técnico programador: no se precisa una preinstalación del programa, la instalación en el equipo del usuario la realizará este mismo asistido por el propio programa.

Por tanto el coste se determinará exclusivamente para las horas empleadas por el programador.

Consultando en el mercado sobre el precio actual de la hora de un programador e ING. Técnicos y realizando la media entre todas tenemos un precio aproximado de:

PROGRAMADOR.....50 Euros. por hora

Estudio de tiempos y coste de la mano de obra

Programación

Se realiza por el técnico programador, se ha calculado que este emplea exclusivamente en la tarea de programación y depuración un total de **100horas**.
Teniendo en cuenta que la estructura del conjunto del programa está compuesto por tres partes bien diferenciadas que son :

Diseño de los gráficos en tres dimensiones.

Integración de la biblioteca gráfica en el programa.

Un manual del usuario

Se calculara los tiempos empleados, que darán el total reseñado anteriormente, desglosándolos en función de cada una de estas.

Tiempos para la integraciñón de QtCreator

TIEMPOS:

Proyecto.pro:

-Procedimientos y funciones de los que consta: 5horas

Glwidge.cpp:

-Procedimientos y funciones de los que consta: 10horas

glwidget.h:

-Procedimientos y funciones de los que consta: 5horas

Proyecto.qrc:

Procedimientos y funciones de los que consta: 5horas

Main.cpp:

Procedimientos y funciones de los que consta: 5horas

Mainwindow.cpp:

Procedimientos y funciones de los que consta: 5horas

Mainwindow.h:

Procedimientos y funciones de los que consta: 5horas

PRESUPUESTO

TIEMPOS: **40 HORAS**

Tiempos para la creación del diseño

en este apartado se incluye las infinitas simulaciones que se ha tenido que realizar para ir mejorando el diseño.

TIEMPOS:

Coche Nissan "Note":

Procedimientos y funciones de los que consta: 20 horas

Carro de golf

Procedimientos y funciones de los que consta: 10 horas

PRESUPUESTO

TIEMPOS: **30 HORAS**

Cuadro resumen de tiempos y costes para la programación

En el cuadro de la página siguiente tenemos un resumen agrupado de las diversas partidas del presupuesto vistas en los apartados anteriores y su desglose

DESCRIPCIÓN		TIEMPOS (HORAS)	COSTES (EUROS)
FASES	INTEGRACIÓN	40	2000
	DISEÑO	30	1500

TOTAL DE INTEGRACIÓN Y DISEÑO: 70 horas

Tiempos para el programa de instalación

PRESUPUESTO

Se trata del programa que se instalará de forma guiada en el ordenador. Se ha empleado un total de **5 horas**.

Tiempos para la realización del manuual del usuario

En el presente presupuesto se repercute como gasto la confección del manual del usuario por considerarse parte imprescindible, junto con el programa.

Se emplean un total de **25 horas**.

Costes directos totales

tiempos integración: 40 horas
tiempos del diseño de ambos vehículos: 30 horas
tiempos para el programa de instalación: 5horas
tiempos para la realización del manual del usuario: 25 horas

TOTAL 100horas
costes directos 100 x 50 = 5000 Euros.

COSTES INDIRECTOS

Los costes indirectos por ser de determinación apriorística, se calcularán considerando un porcentaje estimado en un 12% de los costes directos.

costes indirectos 5000 x 0,12 = 600 Euros

CUADRO RESUMEN DEL PRESUPUESTO

TOTAL DEL PRESUPUESTO DEL PROYECTO	
TOTAL DE LOS COSTES DIRECTOS	5000 Euros
TOTAL DE LOS COSTES INDIRECTOS	600 Euros

RESULTADO TOTAL DE: 5600 Euros.

Glosario

“SGI	*Sillicon Graphics, Inc*.
OPENGL	*Open Graphics Library*
GUI	*Interfaz gráfica de usuario*
IDE	*Entorno de desarrollo integrado*
API	*Application Programming Interface*
GLU	*Graphics Utility Library*

REFERENCIAS

[1]Richard S. Wright and Benjamin Lipchak (2005). "*OpenGl Superbible" [la biblia de opengl]*. Cecilia Poza Melero, EDICIONES ANAYA MULTIMEDIA.). ISBN: 84-415-1794-0.

[2] *"glprogramming"*.[traducida por translate.google. http://translate.google.es/translate]

Disponible [Internet]:

http://glprogramming.com/red/ [1 de agosto del 2011]

[3] Jorge García (Bardok)(2003) . *"Curso de introducción a OpenGL"* ,(v1.0).

[4] *"qt-opengl"* Disponible [Internet]:

http://prezi.com/wafu2knotl0c/qt-opengl/ [1 de agosto del 2011]

[5] Jimenez Ruiz, Manuel. "*Desarrollo de herramientas multimedia para el apoyo del aprendizaje de la librería gráfica OpenGL.",* Proyecto de fin de carrera, Universidad de Valladolid, mayo del 2000.

[6] Cano Utrera, Andres. *"Nuevas Tecnologías de la Programación Modulo 2: Programación grafica en entorno UNIX con una librería de alto nivel (Qt)".* Dpto. Ciencias de la Computación e I.A, Universidad de Granada, Octubre de 2008.

[7] *"Introducing Qt Creator"*.[traducida por translate.google. http://translate.google.es/translate]

Disponible [Internet]:

http://doc.qt.nokia.com/qtcreator-2.3/index.html [1 de agosto del 2011]